Departamento de Informática

PROYECTO FIN DE CARRERA

Aplicación de algoritmos ACO a Planet Wars

Autor: María González Evstrópova

Tutor: Yago Sáez

Leganés, 21 junio de 2015

Título: Aplicación de algoritmos ACO a Planet Wars
Autor: María González Evstrópova
Director: Yago Sáez

EL TRIBUNAL

Presidente: Pedro Isasi

Vocal: Manuel Velasco

Secretario: Dolores Cuadra

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 30 de junio de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Quiero agradecer de todo corazón el apoyo que me han brindado mi pareja y mi familia durante todos estos años, ya que sin ellos no habría sido posible.

Por último, agradecer a mi tutor la paciencia infinita con la que me ha guiado en todo este largo proceso y gracias a cuyo apoyo he conseguido finalizar este proyecto.

Resumen

El objetivo de este proyecto es el estudio y la aplicación de los algoritmos de optimización de colonias de hormigas (ACO) a un problema multiobjetivo complejo presentado en forma de un videojuego llamado Planet Wars, inspirado a su vez en el popular videojuego Galcon. A lo largo de este trabajo se hace un estudio de la problemática planteada, con las características propias de Planet Wars, así como el diseño de la solución desarrollada. Por último, se hace un análisis de los resultados obtenidos, identificando los puntos fuertes y los aspectos de mejora para desarrollos futuros.

Palabras clave: ACO, optimización de colonias de hormigas, Inteligencia artificial, videojuegos, Planet Wars, Galcon, Google AI Challenge.

Abstract

The main objective of this Project is the study and practical application of Ant Colony Optimization algorithms to solving a complex multiobjective task posed as a videogame called Planet Wars, based on a popular videogame known as Galcon. In this document an analysis of the initial situation and problem restrictions, taking into account constraints specific to Planet Wars, is made and the design of the proposed solution is described. Finally, the developed solution is tested and the obtained results are analyzed, focusing on the successful achievements and areas of improvement that may be addressed in the future.

Keywords: ACO, ant colony optimization, Artificial Intelligence, videogames, Plane Wars, Galcon, Google AI challenge.

Índice general

Índice de figuras

Índice de Fórmulas

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

La mayor motivación para haber elegido este proyecto es la propia naturaleza del mismo, que une dos conceptos muy en auge en la sociedad actual: la Inteligencia Artificial y los videojuegos.

Con más de 300 millones de euros recaudados en España (Navas, 2014) (y más de 90 billones de dólares a nivel mundial (Gartner, 2013)) en 2013, la industria de los videojuegos es una de las más potentes industrias de ocio de los últimos años, superando al cine y la música juntos. Es una industria fuerte, estratégica y en expansión. Es fuerte por su volumen de ingresos, y es estratégica porque genera empleo de alta cualificación, y además es una industria en crecimiento, sobre todo gracias la penetración de las tecnologías móviles y los smartphones.

Por si esto fuera poco, la industria de los videojuegos tiene un alto componente de I+D muy ligado a la ingeniería informática en casi todos los eslabones de su cadena de valor: el desarrollo del propio videojuego, fabricación de consolas, periféricos y middleware, e incluso, la publicación y distribución, sobre todo teniendo en cuenta la reciente tendencia a ofrecer el producto final como contenido descargable a través de tiendas virtuales.

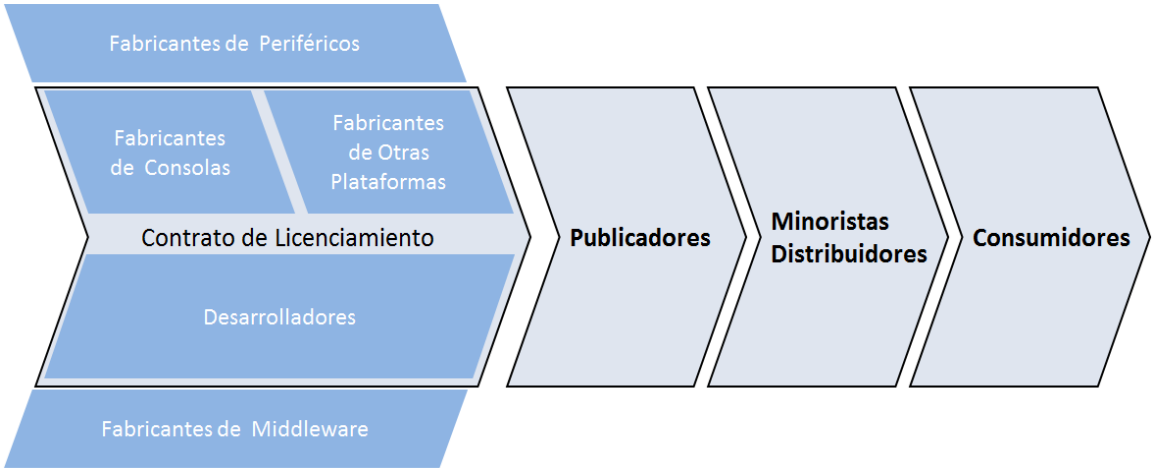


Figura 1: Cadena de valor de la industria de los videojuegos (Raposo, 2008).

El punto de contacto entre los videojuegos y la Inteligencia Artificial se produce en la fase del desarrollo del videojuego, donde la comunidad de videojuegos demanda mayor innovación en IA y la comunidad científica de la IA encuentra un campo de exploración e investigación muy abierto. La Inteligencia Artificial tiene múltiples vertientes, pero aplicada a los videojuegos se centra sobre todo en las técnicas orientadas a producir la ilusión de inteligencia en el entorno del jugador y, principalmente, en los personajes no jugadores (PNJs). De esta forma se crean agentes autónomos que puedan pensar, evaluar y actuar aplicando ciertos principios de coherencia, con la finalidad conseguir que el jugador esté inmerso en la actividad que está ejecutando. Para ello, normalmente se combinan varias técnicas más tradicionales como redes neuronales, algoritmos genéticos, máquinas de estados finitos, etc., consiguiendo comportamientos realistas, desconocidos, divertidos, absurdos… que mejoran la experiencia del juego.

Por último, es importante darse cuenta de que los problemas que se plantean en los videojuegos suelen ser habitualmente representaciones simplificadas de problemas que se pueden presentar en la vida real, aunque en entornos más controlados y con menos variables. En este sentido, la aproximación a su resolución a través de la IA no aporta únicamente el beneficio de una mejor experiencia de juego sino que puede extrapolarse a la vida real, donde la IA puede marcar una gran diferencia en múltiples áreas, tales como los negocios, ingenierías, hospitales, hogares, procesos productivos, etc.

1.2 Objetivos

El principal objetivo de este proyecto es aplicar algoritmos de computación biológica (y más concretamente optimización de colonias de hormigas - ACO) a la resolución de un problema multiobjetivo complejo en un entorno competitivo. El problema a resolver se presenta en forma de un videojuego llamado Planet Wars que consiste en una batalla galáctica, donde dos jugadores luchan por conquistar la galaxia. Ambos contrincantes disponen de unos recursos limitados y una serie de acciones que les permite obtener más recursos, bien del territorio neutral o bien quitándoselos al adversario, con el fin de proclamarse vencedor cuando acabe la partida. Como resultado de este proyecto, se deberá diseñar y desarrollar un agente autónomo que aplique algoritmos ACO para

diseñar estrategias a seguir para obtener recursos colonizando planetas, en función de varios parámetros, y a la vez defenderse de los ataques del contrincante.

En base a ese objetivo principal, se proponen los siguientes objetivos parciales:

- Estudiar el problema a resolver, identificando los distintos objetivos a conseguir para optimizar la solución global.
- Diseñar e implementar un agente autónomo que busque soluciones al problema planteado aplicando algoritmos ACO en un entorno simplificado, sin contrincante o con un contrincante muy sencillo.
- Diseñar e implementar un agente autónomo que busque soluciones al problema planteado aplicando algoritmos ACO en un entorno real, con contrincantes difíciles, desarrollados por otras personas, usando bien algoritmos genéricos o específicos para ese problema, en el marco del concurso Google AI Challenge.

Por último, aunque no por ello menos importante, está el objetivo de afianzar y ampliar los conocimientos adquiridos durante los años de estudio a través de su aplicación a la resolución de un problema más complejo de los que se había enfrentado con anterioridad, con la motivación añadida por la componente competitiva de un concurso.

1.3 Fases del desarrollo

El desarrollo de este proyecto se ha dividido en dos grandes fases.

La primera fase ha consistido en familiarizarse con el problema, las reglas del juego y del concurso, así como el código fuente disponible para empezar a competir con un bot sencillo. A continuación, se ha desarrollado una primera versión del agente, que optimizaba la conquista de los planetas, mientras que la parte defensiva ante adversarios era muy básica. Este agente se probó simulando partidas con los bots iniciales proporcionados por los organizadores del concurso y, finalmente, se presentó al concurso Google AI Challenge, ocupando en la clasificación final la posición 1499 de entre más de 4600 participantes.

La segunda fase se ha centrado analizar las partidas jugadas en el concurso, identificar las principales oportunidades de mejora e implementar las modificaciones necesarias para perfeccionar al agente inicial. Finalmente, se ha probado la versión mejorada del bot en una batería de pruebas, tanto contra los bots básicos como contra aquellos creados por otros concursantes que estuvieran publicados en la página web de Google AI Challenge. Por último, en esta fase se han hecho desarrollos adicionales requeridos para simular el sistema del concurso y el rankeo de participantes de manera similar al original.

1.4 Medios empleados

A lo largo del desarrollo de este proyecto se ha hecho uso de diversos recursos, en función de las tareas específicas que se afrontaban en cada fase: diseño, desarrollo, documentación, etc. Para describirlos con mayor facilidad, se han agrupado los recursos

en varios grupos, dependiendo del tipo y finalidad de cada uno de ellos. Se han identificado las siguientes categorías:

- Herramientas de desarrollo: son todas aquellas herramientas que han hecho posible la labor de desarrollar el agente.
    - NetBeans IDE 7.0
    - Java SE 5 SDK
    - Adobe Flash Player 9.0
- Herramientas de gestión: engloban las herramientas usadas para gestionar las fases del proyecto y generar la documentación de este.
    - Microsoft Word 2007
    - Microsoft Power Point 2007
    - Microsoft Excel 2007
    - Microsoft Project 2007
    - Think Cell 4.1
    - Argo UML Modelling Tool 0.34
- Elementos hardware: agrupan todos los componentes hardware usados en la realización del proyecto:
    - HP Pavilion dv5. Ordenador personal con el que se ha trabajado para el desarrollo del proyecto. Las características básicas de la máquina son:
        - Sistema Operativo: Windows Vista Home Premium.
        - Memoria RAM: 2GB
        - Disco duro: 250 GB.
        - CPU: Intel Core Duo P8600 a 2.40 GHz.
    - Servidor de pruebas y competición de Google AI Challenge: se desconocen las prestaciones de la máquina al recaer su administración en terceros.

1.5 Estructura de la memoria

Para facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada uno de los capítulos que lo componen:

- Capítulo 2: Estado del Arte. En este capítulo se describen brevemente algunos de los algoritmos de IA basada en poblaciones, haciendo hincapié en algoritmos de optimización de colonias de hormigas, así como su aplicación en videojuegos. Finalmente, se incluye un breve resumen de la evolución de la IA dentro del ámbito de los videojuegos, así como su cada vez mayor relevancia y protagonismo a la hora de contribuir al éxito de la industria de los videojuegos en la sociedad actual.
- Capítulo 3: Planet Wars. En este capítulo se realiza un estudio en detalle del problema concreto a resolver y se diseña y formaliza un modelo de agente para resolver cada uno de los subproblemas identificados.
- Capítulos 4: Implementación. En este capítulo se describen los aspectos más relevantes de la fase del desarrollo del agente modelado en el capítulo anterior. Este capítulo contiene el diagrama de clases y la descripción de las principales funcionalidades implementadas del agente.

- Capítulo 5: Validación del modelo: experimentación. En este capítulo se plantea una batería de pruebas para cada uno de los modelos planteados y se presentan y analizan los resultados obtenidos con el fin de validar la solución desarrollada.
- Capítulo 6: Conclusiones y líneas futuras. En este capítulo se plasman las conclusiones finales tras la realización del proyecto en su conjunto, qué conocimientos se han adquirido o afianzado, y qué líneas de desarrollo futuras se plantean en base a los resultados obtenidos.

Capítulo 2

Estado del Arte

Puesto que este proyecto consiste en aplicar algoritmos de Enjambres de Partículas a resolución de un problema que se presenta en forma de videojuego, este capítulo presenta un breve estudio de ambas vertientes y los trabajos más recientes que se han realizado en estos campos.

2.1 Optimización

La Inteligencia de Partículas (Swarm Intelligence) es una rama dentro del amplio abanico de técnicas de Inteligencia Artificial, que han surgido como alternativa a los métodos tradicionales para resolver problemas de optimización de alta complejidad, tanto en espacios continuos como discretos. Para comprender mejor esta primera definición, a continuación se explican más en detalle algunos de los términos que se han utilizado para su construcción.

2.1.1 Problema de Optimización

Un problema de optimización se divide principalmente en tres componentes:

- *Función objetivo*. Es la medida cuantitativa del funcionamiento del sistema que se desea optimizar (maximizar o minimizar). Algunos ejemplos de funciones objetivo son maximizar los beneficios económicos de una serie de inversiones,

minimizar el tiempo de reparto de mercancías, minimizar la desviación de previsiones de llamadas respecto de las reales, maximizar la cantidad de producto fabricado con unos recursos determinados, etc.

- *Variables*. Representan las decisiones que se pueden tomar para afectar el valor de la función objetivo. En el caso de fabricación de un producto, pueden ser las dimensiones del producto o la cantidad de recursos que requieren para ser fabricados. En el caso del tiempo de reparto, las variables pueden ser la cantidad de productos entregados y la distancia a los destinos.
- *Restricciones*. Representan el conjunto de relaciones (expresadas mediante ecuaciones e inecuaciones) que ciertas variables están obligadas a satisfacer. Por ejemplo, para el caso de fabricación de productos, pueden ser las cantidades máximas de recursos disponibles o la relación de cantidades de materias primas a usar para obtener un resultado determinado. En el caso de reparto de mercancías, pueden ser la cantidad de vehículos disponibles, la cantidad de conductores cualificados para distintos tipos de vehículos, etc.

Por lo tanto, resolver un problema de optimización consiste en encontrar el valor que deben tomar las variables para hacer óptima la función objetivo satisfaciendo el conjunto de restricciones. En términos matemáticos, se puede expresar de la siguiente manera:

$$max(min)\, f(x)$$
$$x \in \Omega \subseteq \mathbb{R}^n$$

Fórmula 1: Representación matemática del problema de optimización.

Donde $X = (x_1, \ldots, x_n)$ es un vector y representa las variables de decisión; $f(X)$ es la función objetivo y mide la calidad de la solución formada por las variables de decisión; Ω es el conjunto de restricciones (o decisiones factibles) del problema y se puede representar algunas veces como un sistema de igualdades o desigualdades:

$$g(x_1,\ldots,x_n) \leq 0$$
$$h(x_1,\ldots,x_n) \leq 0$$

Fórmula 2: Sistema de igualdades o desigualdades.

Al vector de variables de decisión X se le llama también el *mínimo global* de f en R^n. Desde un punto de vista práctico, encontrar el mínimo o máximo global de una función no-lineal cualquiera es un problema abierto en matemáticas. Los métodos existentes sólo obtienen mínimos locales. Un punto X es un *mínimo local* de f en R^n si existe $r > 0$ tal que:

$$f(\bar{x}) \leq 0 \ \ para\ ||\bar{x}-x|| \leq r$$

Fórmula 3: Definición de mínimo local

El mínimo global de f será el mínimo local de menor valor de la función objetivo.

2.1.1.1 Optimización Combinatoria

Un tipo de problemas de optimización son aquellos que requieren combinaciones de valores, y se les denomina de optimización combinatoria. La optimización combinatoria es una rama de la optimización en matemáticas aplicadas y en ciencias de la computación, relacionada a la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional, así como otros campos como la inteligencia artificial e ingeniería del software.

Una instancia de un problema de optimización combinatoria puede ser descrita formalmente como una tupla:

$$(X, P, Y, f, extr)$$

Fórmula 4: Definición formal de optimización combinatoria (Optimización combinatoria, 2012)

Donde:

- X es el espacio de soluciones (en el cual f y P están definidos)
- P es la factibilidad del predicado
- Y es el conjunto de soluciones factibles
- f es la función objetivo
- Extr es el extremo (normalmente min o max)

En un problema combinatorio de optimización se desea encontrar un orden específico sobre un conjunto de elementos discretos. Para ilustrar esta definición puede considerarse el problema de encontrar la ruta que debe seguir un viajero para visitar un número determinado de ciudades, de manera que la distancia recorrida sea mínima. En este problema una solución es una posible ruta (orden), y la solución óptima es la ruta que minimiza la distancia recorrida (un orden específico) (Sait, 1999) (Aarts, 2003). Problemas que se ajustan a esta definición aparecen en campos tan diversos como en el diseño de nuevas moléculas, de redes de telecomunicaciones y de nuevas aleaciones, en el posicionamiento de satélites, en la planeación de redes de transmisión de energía y en el desarrollo de circuitos impresos (Grötschel, 1995) (Hoffman, 2000).

2.1.2 Métodos de Optimización

Un algoritmo de optimización, también conocido como técnica de programación matemática, es un método numérico que encuentra un valor $X = (x_1, \ldots, x_n)$ dentro del espacio de búsqueda n-dimensional R^n que maximice o minimice la función $f(X)$, por medio de selección sistemática de valores de la variable X, aplicándole las restricciones del problema. No existe ningún método de optimización que pueda resolver eficientemente todo tipo de problemas y de ahí que se hayan desarrollado diversos métodos a lo largo de los años.

Los métodos de optimización se pueden clasificar en dos grandes vertientes:

- *Clásicos*: Son un amplio abanico de técnicas analíticas que han evolucionado a partir del cálculo matemático y que han servido de punto de partida para los métodos más avanzados de optimización. De forma muy general y aproximada se puede decir que los métodos clásicos buscan y garantizan un óptimo local.

Los métodos tradicionales se pueden agrupar en tres grandes conjuntos:

- Las técnicas de *programación matemática:* son útiles para encontrar el mínimo de una función de varias variables sujeta a un conjunto de restricciones.
- Las *técnicas estocásticas:* pueden usarse para analizar problemas descritos por un conjunto de variables aleatorias con una distribución de probabilidad conocida.
- Los *métodos estadísticos:* nos permiten analizar datos experimentales y construir modelos empíricos para obtener la representación más precisa posible del problema real.

Algunos ejemplos de los métodos clásicos son:

Técnicas de Programación Matemática	Técnicas Estocásticas	Métodos Estadísticos
Métodos de cálculo	Teoría de decisión estadística	Análisis de regresión
Cálculo de variaciones	Procesos de Markov	Análisis de clusters
Programación no lineal	Teoría de colas	Reconocimiento de patrones
Programación geométrica	Teoría de renovación	Diseño de experimentos
Programación cuadrática	Métodos de simulación	Análisis discriminatorio
Programación lineal	Teoría de confiabilidad	
Programación dinámica		
Programación entera		
Programación estocástica		
Programación separable		
Programación multiobjetivo		
Métodos de redes: CPM y PERT		

Tabla 1: Métodos de Optimización Clásicos

- *Metaheurísticos*: son métodos aproximados diseñados para resolver problemas de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. En estos métodos se sacrifica la garantía de encontrar la mejor solución a cambio de obtener una solución buena en un tiempo de computación relativamente reducido. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos (Osman, 1996).
  La metaheurística se puede definir formalmente como un método iterativo que guía a la heurística subordinada, combinando diferentes conceptos para la exploración y explotación de resultados de búsqueda, y usando estrategias de aprendizaje para estructurar la información con el fin de encontrar eficazmente soluciones cercanas a la óptimas.
  La mayor ventaja de los métodos metaheurísticos frente a otros está en su gran flexibilidad, lo que permite usarlos para abordar una amplia gama de problemas.

Entre los metaheurísticos más exitosos se encuentran el recocido simulado (simulated annealing), la búsqueda tabú (tabu search), la búsqueda local iterativa, las redes neuronales artificiales y los algoritmos basados en poblaciones (GA - Algoritmos Genéticos, Optimización por Colonias de Hormigas – ACO, etc.). Estos últimos se explican con más detalle en la siguiente sección, puesto que es en esta categoría donde se engloba el algoritmo usado para resolver el problema que se presenta en este trabajo.

2.2 Algoritmos basados en Poblaciones

Para resolver problemas complejos de optimización combinatoria, los métodos constructivos han evolucionado a métodos de búsqueda local y, finalmente a algoritmos basados en poblaciones. Estos últimos son muy populares actualmente puesto que proveen buenas soluciones al utilizar un método constructivo para la obtención de la población inicial, y una técnica de búsqueda local para mejorar la solución de la población. Además, los métodos basados en poblaciones tienen la ventaja adicional de ser capaces de combinar buenas soluciones con el fin de obtener unas mejores, ya que se considera que las buenas soluciones comparten componentes con las soluciones óptimas. A estos métodos se les conoce como algoritmos de computación evolutiva (Evolutionary Computation – EC).

2.2.1 Computación Evolutiva

La computación evolutiva comprende paradigmas de clasificación y optimización que se apoyan en la combinación de las teorías de la evolución y teorías de la computación. Abarca un conjunto de técnicas iterativas que, dada una función a optimizar, crean aleatoriamente un conjunto de soluciones – individuos o elementos del dominio de esa función – y les aplican la función objetivo en forma de una función de fitness que determina la calidad de la solución. En cada iteración se alternan periodos de auto adaptación, los cuales implican cambios en el individuo, con periodos de cooperación, lo que implica el intercambio de información entre individuos.

Los elementos principales que comparten los algoritmos que se agrupan bajo el paraguas de la Computación Evolutiva son (Muñoz, López, & Caicedo, 2008):

- La existencia de individuos que pueden representar soluciones de un problema. Estas soluciones pueden ser factibles o no, parciales o completas, individuales o grupales.
- Un proceso de evolución que permite definir los cambios en la población a cada generación o de manera continua.
- Una definición de vecindad que permite conocer el modo en el que los individuos intercambian información.
- Un mecanismo que permita identificar las fuentes de información de un individuo.
- Una medida de la factibilidad de la solución obtenida, lo cual permite determinar cuál es buena, óptima o inadecuada.
- Un mecanismo de intensificación, que corresponde al uso de métodos que puedan generar mejoras significativas durante la fase de auto adaptación. Este mecanismo

realiza mejoras sobre un individuo sin tener en cuenta la información suministrada por otros individuos, permitiendo la intensificación de la búsqueda sobre algunas regiones del espacio.

- Un mecanismo de diversificación, el cual permite evitar convergencia hacia puntos óptimos locales. Este procedimiento modifica cada individuo independientemente, pero al contrario del mecanismo de intensificación, tiene resultados inesperados sobre un individuo.

La diferente especificación y combinación de estos elementos forma dos grandes vertientes dentro de la Computación Evolutiva:

- *Algoritmos Evolutivos*: Se basan el concepto de evolución biológica de Darwin. Al igual que en la naturaleza, existen diversas variantes de algoritmos evolutivos, aunque la idea principal que aúnan todos ellos es la misma: dada una población de individuos, la presión del entorno natural causa la selección natural (supervivencia del más apto), que a su vez aumenta la calidad de la población. Dentro de esta rama se pueden citar técnicas como los Algoritmos Genéticos o Programación Genética.
- *Inteligencia de Enjambres*: Es una disciplina dentro de la Inteligencia Artificial que se ocupa de crear sistemas multi-agente inteligentes, inspirándose en el comportamiento colectivo de insectos sociales como las hormigas, abejas, termitas, así como de otras sociedades animales como las bandadas de pájaros o bancos de peces. Aunque los individuos que forman la sociedad son muy sencillos, llegan a realizar labores de gran complejidad cuando operan en conjunto. El comportamiento coordinado de la colonia emerge a partir de acciones o interacciones entre agentes relativamente sencillas.

A continuación, se muestran las principales técnicas dentro de la Computación Evolutiva.

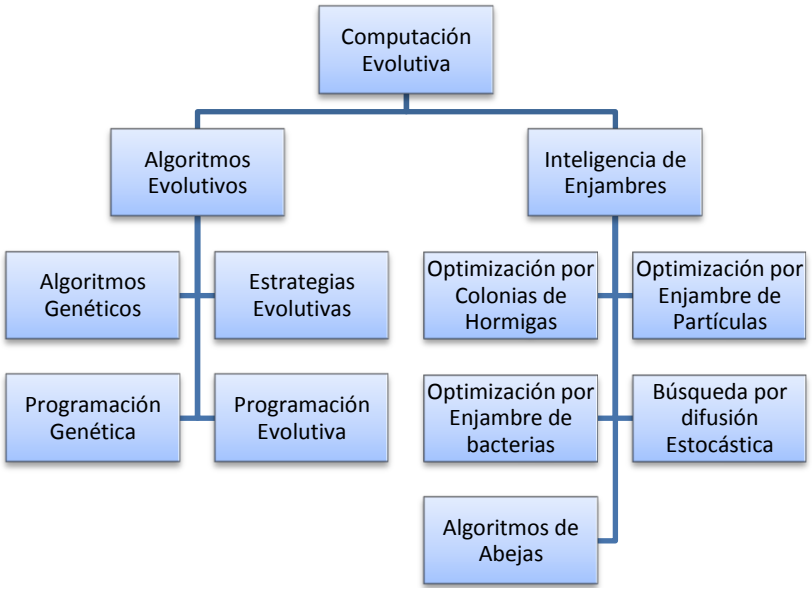


Figura 2: Taxonomía de la computación evolutiva (Muñoz, López, & Caicedo, 2008).

2.2.2 Optimización por Colonias de Hormigas (ACO)

La optimización por colonia de hormigas (ACO) es una familia de algoritmos derivados del trabajo realizado por Dorigo et al., (Dorigo, Maniezzo, & Colorni, 1991), basada en el comportamiento social de las hormigas, las cuales usan una forma de comunicación basada en sustancias químicas denominadas feromonas. Estas sustancias, depositadas por la hormiga al avanzar por un camino, ejercen una acción sobre la decisión de las hormigas precedentes, las cuales escogen el camino que posea una mayor concentración de sustancia, permitiendo que encuentren la ubicación de las fuentes de alimento y el camino más corto hacia ellas así como su nido. Cuando una hormiga pasa por el mismo lugar que otra antes que ella, el rastro de feromona se intensifica; y si no pasa ninguna, con el tiempo se evapora. De ese modo, cuantas más hormigas pasan por un lugar más feromonas tendrá ese camino y más deseable será para otras hormigas, en cambio en los caminos menos transitados las feromonas se van evaporando haciendo que sean menos deseables.

Se ha demostrado que los rastros de feromona permiten lentamente la optimización distribuida en la cual cada agente sencillo realiza una pequeña contribución en la búsqueda de la mejor solución.

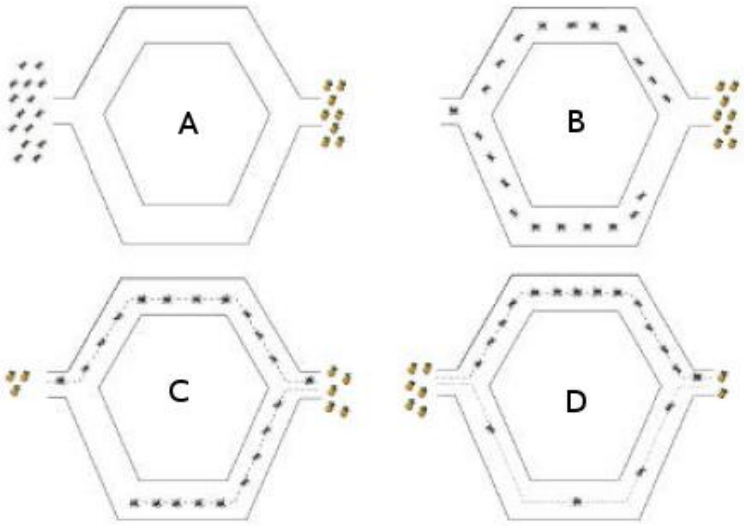


Figura 3: Optimización de distancia por rastro de feromona (Guerra Marrero, 2010)

En los algoritmos de optimización de colonias de hormigas (ACO – Ant Colony Optimization) los rastros químicos de la feromona se representan mediante un modelo probabilístico parametrizado, llamado modelo de feromona (pheromone model). Este modelo consiste en un conjunto de parámetros cuyos valores se denominan valores de feromona y usa una heurística constructiva para desarrollar soluciones de manera probabilística en base a esos valores de feromona. El funcionamiento de los algoritmos ACO se puede resumir en dos grandes pasos:

- Se construyen soluciones usando el modelo de feromona, es decir, una distribución probabilística parametrizada del espacio de soluciones.

- Las soluciones construidas y las posibles soluciones construidas en iteraciones anteriores se usan para modificar los valores de feromona en el modelo con el fin de mejorar la calidad de las futuras soluciones.

El algoritmo ACO se muestra en la siguiente figura:

```
While termination conditions not met do
      ScheduleActivities
            AntBasedSolutionConstruction()
            PheromoneUpdate()
            DaemonActivities(){optional}
      End ScheduleActivities
End while
```

Figura 4: Algoritmo ACO

Este algoritmo se compone de tres partes, que se agrupan en el bloque ScheduleActivities:

- AntBasedSolutionConstruction(): Como se ha mencionado anteriormente, el componente básico de un algoritmo ACO es una heurística constructiva para la composición probabilística del espacio de soluciones. Una heurística constructiva "monta" soluciones a partir de "piezas" que coge del conjunto finito de componentes $\mathfrak{C} = \{c_1, \ldots, c_n\}$. La construcción de la solución comienza con una solución parcial vacía $\mathfrak{s}^P$. A continuación, en cada paso de la construcción, el espacio de soluciones parciales se amplía con un componente factible $\mathfrak{H}(\mathfrak{s}^P)$, definido por el mecanismo de construcción de la solución. El proceso de construcción de la solución se puede ver como un camino en el grafo $\mathfrak{G}_c = (\mathfrak{C}, \mathcal{L})$ cuyos vértices son los componentes de la solución $\mathfrak{C}$ y las aristas pertenecen al conjunto $\mathcal{L}$. Por lo tanto, los caminos permitidos en $\mathfrak{G}_c$ están implícitamente definidos por el mecanismo de construcción de la solución que define el conjunto de componentes factibles $\mathfrak{H}(\mathfrak{s}^P)$ en base a la solución parcial $\mathfrak{s}^P$. La elección de qué componente de $\mathfrak{H}(\mathfrak{s}^P)$ pasará a formar parte de $\mathfrak{s}^P$ en cada paso de la construcción se realiza de manera probabilística siguiendo el modelo de feromona, que define unos parámetros de feromona τ_i asociados a los componentes $c_i \in \mathfrak{C}$. El conjunto de todos los valores de feromonas para todos los componentes se denomina $\mathfrak{I}$. En la mayoría de los algoritmos ACO las probabilidades de escoger un determinado componente para la solución parcial en cada paso, también llamadas probabilidades de transición, se definen como sigue:

$$a_{ij} = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in Ni}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \forall j \in N_i$$

Fórmula 5: Probabilidad de transición en algoritmos ACO

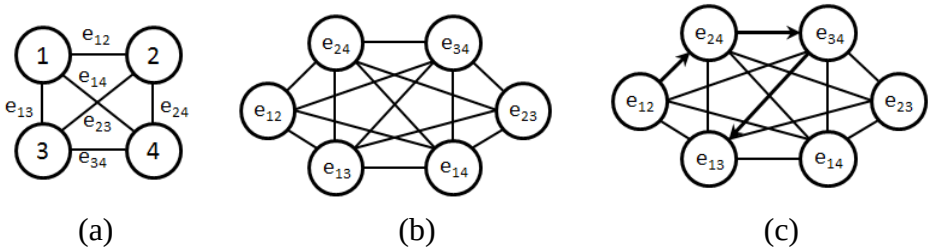
Donde η es una función que asigna pesos a cada componente de la solución $c_j \in \mathfrak{H}(\mathfrak{s}^P)$ en cada paso, dependiendo de la solución parcial existente en cada momento. El peso asignado a cada componente se rige por una función heurística $\eta(c_j)$. Los valores α y β son parámetros positivos que determinan la importancia relativa entre el peso de los parámetros de feromona y los heurísticos.

Si α=0, el algoritmo se comporta como un algoritmo estocástico clásico. Si, por el contrario, β=0, el ACO únicamente se rige por la feromona, presentando una convergencia rápida y estancamiento en una solución subóptima, en la mayoría de los casos.
La probabilidad $p^k_{i,j}(t)$ con la que una hormiga que se encuentra en el componente i, en la iteración t, escogerá el componente j en el próximo paso es:

$$p_{ij}^k(t) = \frac{a_{ij}(t)}{\sum_{l \in N_i^k} a_{il}(t)}$$

Fórmula 6: Probabilidad de transición individual en algoritmos ACO

En la siguiente figura se describen algunos ejemplos de construcción de la solución:



(a) (b) (c)

Figura 5: Ejemplo de construcción de la solución en ACO

(a)Muestra el grafo $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ de un problema de TSP de 4 ciudades. Las aristas e_{ij} que conectan las ciudades tienen asociadas las distancias d_{ij}. El conjunto de componentes para construir la solución $\mathfrak{C}$ consiste de 6 caminos (aristas) que conectan las 4 ciudades. Por lo tanto, el modelo de feromona asigna un parámetro de feromona τ_{ij} para cada arista e_{ij}. Como información heurística, se ha elegido el inverso de la distancia entre dos ciudades, de manera que $\eta(e_{ij}) = 1/d_{ij}$ para cada e_{ij}. A su vez, en el primer paso de la construcción de la solución puede escogerse cualquier componente de solución definido. Para los siguientes pasos, el conjunto de componentes a ser elegidos se restringe de tal manera que la secuencia de los componentes en la solución siempre sigue un camino definido en el grafo $\mathfrak{G}$. (b) Muestra el grafo de construcción de solución $\mathfrak{G}_c = (\mathfrak{C}, \mathcal{L})$. (c) Muestra un camino en el grafo que corresponde a la construcción de la solución $s = (e_{12}, e_{24}, e_{34}, e_{13})$.

- PheromoneUpdate(): Los algoritmos ACO pueden actualizar la feromona de diferentes maneras. Sin embargo, existe un tipo de actualización más extendido, que presentan casi todos los algoritmos ACO por igual. Este método consiste de dos pasos. Primero, se realiza la evaporación de feromona, que disminuye uniformemente todos los valores de feromona. Este paso ayuda a evitar una convergencia prematura del algoritmo hacia un óptimo local, implementando una forma útil de "olvidar" para favorecer la exploración de nuevas soluciones en el espacio de búsqueda. A continuación, se incrementa el valor de feromona de componentes que forman parte de soluciones generadas en esta y/o anteriores iteraciones.
  Como ejemplo, a continuación se muestra la regla de actualización de feromona del algoritmo AS (Ant System), que fue el primer algoritmo ACO planteado:

$$\tau_i \leftarrow (1-\rho) * \tau_i + \rho * \sum_{s \in \mathfrak{S}_{iter} | c_i \in s\}} F(s) \quad \text{para} \quad i=1,\ldots,n$$

Fórmula 7: Regla AS de actualización de feromona.

Donde S_{iter} es el conjunto de soluciones que se generan en la iteración en curso, $\rho \in (0,1]$ indica la tasa de evaporación de feromona y $F: S \rightarrow \mathbb{R}^+$ es una función tal que $f(s) < f(s') \Rightarrow F(s) \geq F(s') \; \forall s \neq s' \in \mathfrak{S}$, también denominada función de fitness o de calidad.
Otros tipos de actualización de feromona son opcionales y se concentran, sobre todo, en la intensificación o diversificación del proceso de búsqueda. Un ejemplo sería disminuir el valor de la feromona en el momento de construir la solución, al añadir el componente c_i a la solución parcial s^P. En este ejemplo, la evaporación inmediata de la feromona favorecería la diversificación del proceso de búsqueda.

- DaemonActions(): estas acciones pueden implementar acciones centralizadas que no pueden llevarse a cabo por las hormigas. Por ejemplo, se podría aplicar un método de búsqueda local para las soluciones parciales construidas, o recoger información global para evaluar la conveniencia de aumentar o disminuir la feromona para influir en el proceso de búsqueda desde una perspectiva global y no local.

Actualmente, dos de las versiones más usadas de algoritmos ACO son el Sistema de Colonias de Hormigas (ACS - Ant Colony System) y Sistema de Hormigas MAX-MIN (MMAS - Max – MIN Ant System), que se describen brevemente en la siguiente sección.

2.2.2.1 MAX – MIN Ant System (MMAS)

En algoritmos MMAS los valores de feromona se acotan en el intervalo $[\tau_{min}, \tau_{max}]$, donde $0 < \tau_{min} < \tau_{max}$. Estos límites explícitos de los valores de feromona impiden que la probabilidad de construir una solución supere un cierto valor mayor que 0, por lo que la posibilidad de encontrar el óptimo global nunca desaparece.

Además, si el algoritmo detecta que el proceso de búsqueda se centra exclusivamente en un subespacio de búsqueda, el proceso se reinicia, volviendo a poner todos los valores de feromona a sus valores iniciales.

Por último, la actualización de feromona se realiza siempre con el valor de la mejor solución ya sea de la iteración, desde el último reinicio o desde el comienzo de ejecución del algoritmo.

2.2.2.2 Ant Colony System (ACS)

El algoritmo ACS se basa en el algoritmo AS, pero introduce varias mejoras importantes. En primer lugar, tras cada iteración, la feromona se actualiza usando la mejor solución encontrada hasta el momento (desde el inicio del comienzo de ejecución del algoritmo). La evaporación de la feromona también se aplica a las componentes que forman parte de la mejor solución hasta el momento.

En segundo lugar, las probabilidades de transición se definen por una regla pseudo-aleatorio-proporcional (pseudo-random-proportional). Usando esta regla, algunos pasos de la construcción de la solución se realizan de manera determinista, mientras que en otros, las probabilidades de transición se calculan según está definido en la Fórmula 6.

Finalmente, durante la construcción de la solución, se decrementa levemente la feromona de cada componente que forma parte de dicha solución.

La siguiente figura muestra el pseudocódigo correspondiente a un algoritmo ACS aplicado a un problema de ejemplo, consistente en minimizar el coste de una función. En cada iteración, dicho algoritmo usa tanto la información histórica (feromona) como heurística para construir la solución paso a paso de manera incremental.

```
Input: ProblemSize, Population_size, m, ρ, β, σ, q_0
Output: Pbest
Pbest ← CreateHeuristicSolution(ProblemSize);
Pbest_cost ← Cost(S_n);
Pheromone_init ← 1.0/(ProblemSize* Pbest_cost);
Pheromone ← InitializePheromone(Pheromone_init);
While ¬StopCondition() do
      For i=1 to m do
            Si ← ConstructSolution(Pheromone,ProblemSize, β, q_0);
            Si_cost ← Cost(Si);
            If Si_cost ≤ Pbest_cost then
                  Pbest_cost ← Si_cost;
                  Pbest ← Si;
            End
            LocalUpdateAndDecayPheromone(Pheromone,Si,Si_cost,σ);
      End
      GlobalUpdateAndDecayPheromone(Pheromone,Pbest,Pbest_cost, ρ);
End
Return Pbest;
```

Figura 6: Pseudocódigo para el algoritmo ACS.

En la mayoría de los problemas combinatorios, un componente i sólo puede ser elegido para incluirse en la solución si aún no forma parte de la misma, y su probabilidad de ser seleccionado es:

$$P_{i,j} \leftarrow \frac{\tau_{i,j}^{\alpha} * \eta_{i,j}^{\beta}}{\sum_{K=1}^{c} \tau_{i,k}^{\alpha} * \eta_{i,k}^{\beta}}$$

Fórmula 8: Probabilidad de transición en algoritmo ACS

Donde $\eta_{i,j}$ es el peso del componente, dado por su función heurística (por ejemplo, *1.0/distance*$_{i,j}$ en el caso del Problema del Viajante - TSP), β es el coeficiente heurístico (normalmente 1.0), $\tau_{i,j}$ es el valor de feromona para el componente, α es el coeficiente histórico y c es el conjunto de componentes seleccionables. El factor *q0* se usa para influenciar bien la elección probabilística del componente o bien la elección directamente del mejor componente disponible.

Para cada solución construida, se aplica la regla de la evaporación local de feromona, según muestra la siguiente fórmula, para disuadir a las siguientes hormigas de que usen

los mismos componentes en el mismo orden y así potenciar la exploración del espacio de búsqueda.

$$\tau_{i,j} \leftarrow (1-\sigma) * \tau_{i,j} + \sigma * \tau^{0}_{i,j}$$

Fórmula 9: Regla de evaporación local de feromona

Donde $\tau_{i,j}$ representa la feromona para el componente, o arista del grafo, *(i,j)*, σ es el factor de evaporación y $\tau^{0}_{i,j}$ es el valor inicial de feromona.

Al final de cada iteración, la feromona se actualiza usando la mejor solución candidata encontrada (en la iteración, desde el inicio de ejecución, etc.), usando la regla que se muestra a continuación:

$$\tau_{i,j} \leftarrow (1-\rho) * \tau_{i,j} + \rho * \Delta\tau_{i,j}$$

Fórmula 10: Regla de actualización global de feromona

Donde $\tau_{i,j}$ es el valor de feromona para el componente, o arista del grafo, *(i,j)*, ρ es el factor de actualización de feromona y $\Delta\tau_{i,j}$ es el coste de la mejor solución global encontrada hasta el momento si el componente *i,j* forma parte de dicha solución o *0* en caso contrario.

2.2.2.3 Aplicaciones

La optimización por colonias de hormigas se diseñó para su uso en problemas combinatorios, como el problema del viajante, el problema de la mochila, la asignación cuadrática, etc.

Problema	Autores	Año	Algoritmo
Problema de viajante	Dorigo, Maniezzo & Colorni Gambardella & Dorigo Dorigo & Gambardella Stützle & Hoos Bullnheimer, Hartl & Strauss	1991 1995 1996 1997 1997	AS ANT-Q ACS & ACS-3-opt MMAS AS $_{rank}$
Asignación cuadrática	Maniezzo, Colorni & Dorigo Gambardella, Taillard & Dorigo Stützle & Hoos Maniezzo & Colorni Maniezzo	1994 1997 1998 1998 1998	AS-QAP HAS-QAP[a] MMAS-QAP AS-QAP[b] ANTS-QAP
Enrutamiento de vehículos	Bullnheimer, Hartl & Strauss Gambardella, Taillard & Agazzi	1996 1999	AS-VRP HAS-VRP
Enrutamiento de red orientado a conexión	Schoonderwoerd, Holland, Bruten & Rothkrantz White, Pagurek & Oppacher Di Caro & Dorigo Bonabeau, Henaux, Guérin, Snyers, Kuntz & Théraulaz	1996 1998 1998 1998	ABC ASGA AntNet-FS ABC-smart ants
Enrutamiento de red sin conexión	Di Caro & Dorigo Subramanian, Druschel & Chen Heusse, Guérin, Snyers & Kuntz Van der Put & Rothkrantz	1997 1997 1998 1998	Ant Net & AntNet FA Regular Ants CAF ABC-backward
Ordenación secuencial	Gambardella & Dorigo	1997	HAS-SOP
Coloración de grafos	Costa & Hertz	1997	ANTCOL

Supersecuencia común más corta	Michel & Middendorf	1998	AS-SCS

Tabla 2: Aplicación de optimización de colonias de hormigas. (Dorigo & Stützle, 2004)

En esta sección se describen brevemente algunos de los problemas más estudiados y su resolución con los algoritmos de colonias de hormigas.

El problema del viajante (TSP)

El problema del viajante, también conocido como Travelling Salesman Problem (TSP) juega un papel muy importante en el desarrollo de los algoritmos ACO porque fue uno de los primeros problemas en ser "atacado" por esos algoritmos, por múltiples razones: es un problema para el que el enfoque de colonias de hormigas es fácilmente adaptable; es uno de los problemas NP – difícil más estudiados en optimización combinatoria; y es fácilmente explicable, de modo que su funcionamiento no requiere demasiadas explicaciones técnicas.

El TSP puede definirse como sigue. Sea C un conjunto de componentes que representan ciudades, L un conjunto de conexiones que unen dichas ciudades y $J_{ci,cj}$ el coste (longitud) de la conexión entre c_i y c_j. TSP plantea el problema de encontrar el circuito Hamiltoniano de longitud mínima en el grafo $G = (G,L)$, donde dicho circuito es un camino cerrado ψ que pasa una y sólo una vez por todos los nodos N_c de G, y su longitud corresponde a la suma de longitudes $J_{ci,cj}$ de todos los arcos de los que se compone. No es necesario que el grafo sea simétrico (en un TSP asimétrico $J_{ci,cj}$ puede ser diferente de $J_{cj,ci}$) ni tampoco completamente conectado (los arcos inexistentes pueden simularse asignándoles un coste infinito).

El procedimiento usado por el algoritmo ACS para resolver este problema comienza por colocar $m \leq N_c$ hormigas repartidos en m ciudades. El estado inicial de la hormiga, es decir, la ciudad de la que parte, puede elegirse aleatoriamente. La memoria $\mathcal{M}^k$ de cada hormiga se inicializa añadiéndole la ciudad de inicio de la misma. A continuación, la hormiga entra en un ciclo que dura N_c iteraciones, en otras palabras, hasta que cada hormiga visite todas las ciudades.

En cada paso, la hormiga parada en el nodo i calcula las probabilidades de transición a las demás ciudades, aplica la regla de decisión sobre el conjunto factible de ciudades vecinas, elige la siguiente ciudad destino, se traslada a dicha ciudad y actualiza su memoria. Una vez realizado el camino completo, lo que ocurre simultáneamente si en cada iteración cada hormiga visita una ciudad, las hormigas usan la información del recorrido guardada en su memoria para evaluar la solución obtenida y actualizar la feromona de los arcos visitados. De este modo se consigue aumentar la deseabilidad del camino más visitado para las futuras hormigas. Por último, la hormiga muere.

En algoritmos AS, todas las hormigas depositan feromona en sus caminos y la evaporación de feromona se produce una vez toda las hormigas hayan acabado su recorrido. La mayoría de los algoritmos ACO desarrollados posteriormente añaden una serie de rutinas de optimizaciones locales, específicas del problema.

La feromona $\tau_{i,j}(t)$ depositada en una conexión $l_{i,j}$ sirve para indicar la deseabilidad de ir a la ciudad j estando en la ciudad i, lo cual se corresponde con la deseabilidad de incluir el arco $l_{i,j}$ en la solución que está siendo construida. El rastro de feromona en los arcos cambia a lo largo del problema, reflejando la experiencia o conocimiento adquirido por las hormigas durante la ejecución del algoritmo. Asimismo, las hormigas depositan una cantidad de feromona proporcional a la calidad de la solución ψ que producen, de modo

que, en este caso, los caminos más cortos tendrán mayor cantidad de feromona. Esto ayuda a dirigir el proceso de búsqueda hacia las soluciones buenas.

La memoria (o estado interno) M^k de cada hormiga contiene la lista de ciudades que ya han sido visitadas hasta el momento y se llama también la lista *tabú*. Dicha lista se usa para representar, para cada hormiga, qué nodos *Cj* todavía están disponibles para visitarse, estando en el nodo *Ci*. Además, una vez completado el camino, la memoria M^k se usa para retroceder sobre sus pasos, pasando por los nodos visitados y depositando feromona en sus arcos.

Enrutamiento de redes

El problema general de encaminamiento de redes puede definirse como el problema de construir y aplicar tablas de enrutamiento para dirigir paquetes de datos a través de nodos de red, consiguiendo maximizar el rendimiento de alguna medida concreta de dicha red.

De manera formal, este problema se puede definir de la siguiente manera. Sean *C* y *L* el conjunto de nodos y enlaces de comunicación de la red, respectivamente. Sea *G=(C,L)* un grafo dirigido donde cada nodo representa un nodo de la red y cada arco orientado, un enlace de transmisión. Cada enlace tiene asociado un coste en función de sus características físicas y el flujo de tráfico. Las aplicaciones que usan las redes de comunicación generan tráfico de datos desde nodos origen hasta nodos destino. En cada nodo, el componente de encaminamiento usa la tabla de enrutamiento local para elegir el mejor enlace de transmisión por el que debe salir el paquete de datos para llegar a su destino. La tabla de enrutamiento $R_i=[r_{ijd}]$ de un nodo cualquiera *i* indica al paquete de datos que entra en nodo *i* y se dirige hacia el nodo *d* cuál debe ser el siguiente nodo $j \in N_i$ al que debe dirigirse, siendo N_i el conjunto de nodos vecinos de *i*. Las tablas de encaminamiento son bidimensionales porque la elección del nodo vecino al que redirigir el paquete de datos entrante se realiza en función del nodo destino final. Las tablas de enrutamiento $A_i=[a_{ijd}]$ que usan los algoritmos de colonias de hormigas conservan esta estructura bidimensional: los rastros de feromona asociados a cada enlace son vectores de N_C-1 posiciones. Estos valores vectoriales de la feromona son la extensión natural de los valores escalares usados en TSP. Los valores de dichas tablas se asignan en función de valores heurísticos y de feromona locales:

$$a_{ijd}(t) = \frac{w * \tau_{ijd}(t) + (1-w) * \eta_{ij}}{w + (1-w)(|N_i| - 1)}$$

Fórmula 11: Asignación de valores en la tabla de enrutamiento.

Donde $j \in N_i$, d es el nodo destino, $w \in [0,1]$ es un coeficiente de peso y el denominador es el factor de normalización.

Este algoritmo ACO presenta otras dos diferencias importantes con respecto a la implementación del TSP. Primero, a cada hormiga se le asigna un par de valores origen-destino *(s,d)* iniciales. Buscando el mejor camino entre *s* y *d*, la hormiga descubre únicamente una parte de la solución global, definida como el conjunto de caminos entre todos los pares de nodos *(i,j)* de la red. Segundo, el coste de cada enlace no se asigna de manera estática, sino que depende de las propiedades de la propia conexión y la carga de tráfico que se presenta en este momento.

En el algoritmo S-AntNet cada hormiga busca el camino mínimo entre un par de nodos de la red. Cada hormiga parte de un nodo origen *s* hacia un nodo destino *d*,

seleccionados según los patrones de circulación de datos habituales de dicha red, y va "saltando" de nodo en nodo hasta que alcance su meta. Estando en el nodo *i*, la hormiga *k* elige el siguiente nodo de manera probabilística usando una regla de decisión en función su memoria M^k y la tabla de enrutamiento A_i. Si $M^k \subseteq N_i$, es decir, si todavía queda algún nodo vecino no visitado, entonces la probabilidad de transición es:

$$p_{ijd}(t) = \begin{cases} a_{ijd}(t) & if\ j \notin \mathrm{M}^k \\ 0 & if\ j \in \mathrm{M}^k \end{cases}$$

Fórmula 12: Regla de transición en S-AntNet para $M^k \subseteq N_i$

Si por el contrario, ya se han visitado todos los nodos vecinos, la hormiga escogerá un nodo j ∈ Ni con probabilidad uniforme:

$$p_{ijd}(t) = {}^{1}/_{(n-1)}$$

Fórmula 13: Regla de transición en S-AntNet para $M^k \nsubseteq N_i$

Las hormigas tratan de evitar hacer bucles (Fórmula 12), pero en caso de que ya haya pasado por todos los nodos vecinos del nodo *i*, la hormiga no tiene más remedio que volver por segunda vez a uno de los nodos visitados, generando así un bucle. Si esto sucede, el recorrido del bucle se borra de la memoria de la hormiga. Considerando el comportamiento estocástico del algoritmo, las probabilidades que esa misma hormiga vuelva a formar el mismo ciclo son reducidas.

S-AntNet sigue depositando la feromona en los arcos, pero asociada a pares arco-destino. En otras palabras, cada arco *(i,j)* tiene *n-1* rastros de feromona $\tau_{ijd} \in [0,1]$ asociados, uno para cada posible nodo destino *d* alcanzable por la hormigas desde el nodo *i*. Cada arco además tiene asociado un valor heurístico $\eta_{ij} \in [0,1]$ independiente del nodo destino:

$$\eta_{ij} = \frac{q_{ij}}{\sum_{l \in N_i} q_{il}}$$

Fórmula 14: Heurística en S-AntNet

Donde q_{ij} es el tamaño de la cola del enlace del nodo *i* hacia el nodo *j*, medido en bits esperando a ser enviados.

La hormiga se mueve por los mismos enlaces de transmisión que los datos que se envían, por lo que experimentará los mismos retrasos de la red. Por lo tanto, el tiempo T_{sd} que una hormiga tarda en llegar del nodo origen *s* al destino *d* puede ser un buen indicador de la calidad del trayecto realizado por la hormiga. Sin embargo, la calidad total del camino debe tener en cuenta también el estado de la red en un momento dado, puesto que un camino más lento puede ser una mala solución cuando hay poco tráfico de datos y sí pasar a ser una buena opción cuando el grado de congestión de red aumenta.

Una vez que una hormiga k ha alcanzado su destino, retrocede sobre sus pasos y actualiza la feromona τ_{ijd} de cada arco visitado:

$$\tau_{ijd}(t) \leftarrow \tau_{ijd}(t) + \Delta\tau^{\mathrm{k}}(\mathrm{t})$$

Fórmula 15: Regla de actualización de feromona en S-AntNet.

Tras actualizar la feromona en los enlaces recorridos, se procede a la evaporación de feromona de los arcos salientes del nodo *i* hacia el destino *d*, según la siguiente regla:

$$\tau_{ijd}(t) \leftarrow \frac{\tau_{ijd}(t)}{(1+\Delta\tau^{k}(t))} \; \forall j \in N_i$$

Fórmula 16: regla de evaporación de feromona en S-AntNet

Donde N_i es el conjunto de nodos vecinos del nodo *i*.

Asignación cuadrática

Sea n un conjunto de actividades que deben realizarse en n localizaciones. Las distancias entre dichas localizaciones se indican en la matriz $D=[d_{ij}]$, donde d_{ij} es la distancia Euclídea entre localizaciones *i* y *j*. Sea $F = [f_{hk}]$ la matriz que define el flujo de información (materiales, personal, etc.) entre actividades, donde f_{hk} es el flujo entre actividades *h* y *k*. Una asignación es una permutación π de $\{1,\ldots,n\}$ donde $\pi(i)$ es una actividad que se asigna a la localización *i*. El problema consiste en encontrar la permutación π_m tal que el producto del flujo de información por la distancia entre localizaciones sea mínimo.

Cabe señalar que el TSP puede verse como una caso particular del problema de asignación cuadrática (QAP – Quadratic Assignment Problem), donde las actividades serían números enteros de *1* a *n* y las localizaciones – las ciudades. El TSP consistiría en asignar los números a las ciudades de tal manera que el camino que recorre las ciudades en el orden dado por esos números sea mínimo.

Usando la nomenclatura de algoritmos ACO, el QAP se plantea de la siguiente manera. Sea C el conjunto compuesto de actividades y localizaciones. Las transiciones se realizan entre actividades y localizaciones y viceversa. Normalmente, una hormiga comenzará a construir una solución eligiendo una actividad y una localización a la que asignar esa actividad. A continuación, se repite el proceso hasta que todas las actividades hayan sido asignadas. Tanto las actividades como las localizaciones se eligen del conjunto de vecinos factibles, es decir, de actividades y localizaciones todavía no asignadas. Habitualmente, los rastros de feromona se asocian a transiciones entre actividades y localizaciones (es decir, la elección de qué localización escoger para una actividad determinada) y no viceversa. Sin embargo, no existe ningún impedimento formal en definir un algoritmo ACO donde las transiciones entre localizaciones y actividades vayan en función de rastros de feromona.

Supersecuencia común más corta

Dado un conjunto *L* de cadenas sobre un alfabeto Σ, el problema de la supersecuencia común más corta consiste en encontrar una cadena de longitud mínima que sea una supersecuencia de todas las cadenas en *L*. Una cadena *B* es supersecuencia de otra cadena *A* si *B* se puede obtener a partir de *A* insertando en *A* cero o más caracteres. Por ejemplo, sea *L = bbbaaa, bbaaab, cbaab, cbaaa*. La cadena *cbbbaaab* sería la supersecuencia común más corta para *L*. Las hormigas construyen la solución quitando iterativamente caracteres del inicio de las cadenas contenidas en L y añadiéndoselos a la supersecuencia en construcción. Cada hormiga mantiene un vector de punteros a los comienzos de las cadenas (siendo el comienzo el primer carácter no eliminado de la cadena) y se mueven en el espacio de vectores factibles. Las transiciones se definen

implícitamente por las reglas que indican de qué manera se pueden quitar los caracteres del inicio de una cadena. El conjunto de restricciones, a su vez, se definen implícitamente por el orden de los caracteres en las cadenas.

Problema de encaminamiento de vehículos

El problema de encaminamiento de vehículos o VRP – Vehicle Routing Problem consiste en dar servicio a un conjunto de clientes usando un conjunto de vehículos disponibles, minimizando algún indicador de coste (tiempo, recursos, distancia, etc.) y cumpliendo una serie de restricciones. Las características de los vehículos y las restricciones determinan el tipo particular de VRP que se intenta resolver.

Un ejemplo sencillo se puede definir de la siguiente manera. Sea $G = (V,A,d)$ un grafo completo dirigido, donde $V = \{v_0,\ldots,v_n\}$ es el conjunto de vehículos, $A = (i,j) : i \neq j$ es el conjunto de arcos y $d_{ij} \geq 0$ es el peso asociado al arco (i,j) y representa la distancia entre v_i y v_j. El vértice v_0 representa el garaje y los demás vértices representan las localizaciones de los clientes. Cada cliente v_i tiene asociado un pedido $d_i \geq 0$ y un tiempo de servicio $\theta_i \geq 0$, siendo $d_0 = 0$ y $\theta_0 = 0$. El objetivo es encontrar un conjunto de rutas de coste mínimo de tal manera que:

- cada cliente sea visitado por un sólo vehículo,
- los pedidos a transportar por un sólo vehículo no superen su capacidad D,
- todos los vehículos comienzan y terminan su recorrido en el garaje y
- el camino total recorrido por cada vehículo no supere una longitud total L

Se puede observar que los problemas VRPs y TSPs están estrechamente relacionados: un VRP consiste en múltiples TSPs que comienzan y acaban en una ciudad común. Al igual que en el TSP, las hormigas construyen la solución visitando secuencialmente todas las ciudades. Los componentes de la solución son ciudades y las transiciones son los arcos entre ellas, a los que se asocia un rastro de feromona. El conjunto de vecinos factibles se compone de localizaciones vecinas que no hayan sido visitadas aún.

Problema multidimensional de la mochila

El problema multidimensional de la mochila (MKP – Multidimensional Knapsack Problem) puede plantearse como un problema de asignación de recursos, en el que se dispone de I = {1,…,m} recursos (mochilas) y J={1,…,n} objetos tal que el objeto j aporta un beneficio p_j. Cada recurso tiene una capacidad c_i y cada objeto j consume r_{ij} cantidad de recurso i (por ejemplo, espacio). El objetivo es encontrar un subconjunto de elementos que maximice el beneficio total, mientras que cumple con una serie de restricciones establecidas:

$$\text{Maximizar:} \quad \sum_{j=1}^{n} p_j..x_j$$

$$\sum_{j=1}^{n} r_{ij}..x_{ij} \leq b_i \forall i \in 1..m$$

$$\text{Sujeto a:} \quad x_j \in \{0,1\} \; \forall j \in 1..n$$

Fórmula 17: Definición formal de MKP

Donde x_j es una variable que indica si un objeto ha sido seleccionado para un recurso ($x_j = 1$) o no ($x_j = 0$).

En el grafo del MKP, los nodos se corresponden con los objetos, que están conectados con todos los demás nodos. El hecho de ser un grafo completamente conectado supone que, tras elegir un objeto i, cualquier objeto j puede ser elegido si hay recursos disponibles suficientes y el objeto j todavía no ha sido elegido. Las hormigas comienzan a construir la solución eligiendo al azar el primer componente de la solución, a añadiendo más componentes de manera iterativa, cumpliendo siempre las restricciones del problema. Una vez que todas las hormigas han construido una solución, deben actualizar la feromona premiando las mejores soluciones. La feromona se puede actualizar de tres maneras posibles:

- La primera posibilidad es depositar feromona en los objetos seleccionados, de tal manera que al aumentar la deseabilidad del objeto j, éste tenga más probabilidad de ser seleccionado para consumir el recurso.
- La segunda posibilidad es depositar feromona en pares de objetos j, j+1 de tal manera que aumente la deseabilidad de escoger el objeto j+1 si el último objeto seleccionado fue el objeto j.
- La última posibilidad consiste en depositar feromona en todos los pares de objetos (j,k), de tal manera que la deseabilidad de escoger el par completo de objetos en la siguiente iteración aumente si la solución encontrada en la presente iteración ya contiene alguno de los objetos de ese par.

Se dan *m* restricciones en el problema, por lo que el MKP se denomina también Problema de la Mochila m-dimensional.

2.3 Aplicaciones en Videojuegos

La Inteligencia Artificial desarrolló una vertiente ligada a videojuegos desde el comienzo de los mismos en los 1970. Tuvo un origen humilde, y su percepción por el público fue extremadamente influenciada por los juegos simplistas de los 70 y los 80. La propia industria de los videojuegos ha tomado pocas medidas para cambiar esta percepción, hasta hace algunos años ligada a los famosos Inky, Pinky, Blinky y Clyde que aparecieron con el *Pac-Man*. Y es que, en sus orígenes, la IA de videojuegos fue diseñada para juegos de máquinas recreativas, con el fin de asegurar que los jugadores sigan echando monedas. Juegos como *Pong, Pac-Man, Space Invaders, Donkey Kong* y *Joust* usaban una serie de reglas sencillas y secuencias de acciones pregrabadas, combinada con alguna toma de decisiones aleatorias, para que el comportamiento sea menos predecible. Por ejemplo, el Pong, usaba en sus primeras versiones una IA muy rudimentaria, que consistía en que el oponente moviera la raqueta en función de la posición de la bola, sin poder superar el límite de velocidad máxima impuesto.

Los juegos de ajedrez fueron los primeros juegos con un fuerte componente de IA. No es de extrañar que juegos como el *Chessmaster 2000* presenten unos formidables rivales con IA, basados inevitablemente en búsqueda de árbol.

Los juegos de estrategia fueron los siguientes en aliarse con IA, puesto que no pueden llegar muy lejos usando sólo los componentes gráficos y requieren una fuerte dosis de IA para ser al menos aceptables. Se pueden destacar *Civilization* y *Civilization 2* como algunos de los primeros ejemplos.

Sin embargo, en los últimos años ha comenzado la revolución de la IA en la industria de los videojuegos. A medida que mejora el hardware de renderizado 3D y la asombrosa

calidad gráfica de los videojuegos se aproxima a un punto en el que su rentabilidad comienza a disminuir, la IA surge como un factor de éxito de los videojuegos modernos, marcando la diferencia entre pasar desapercibidos o convertirse en un bestseller.

Juegos como el *Warcraft II* de *Blizzard* o *Age of Empires 2: The Age of Kings* de *Ensemble* presentan algunos de los primeros componentes de inteligencia artificial de estrategia en tiempo real (RTS - Real-time strategy) competentes, ofreciendo oponentes que suponen nuevos retos a los jugadores. Estos módulos destacan incluso más, teniendo en cuenta que deben cumplir con unos estrictos requisitos de rendimiento en tiempo real, como las búsquedas de ruta para cientos de unidades a la vez en un momento dado.

En el campo de los first person shooters, cabe mencionar varios títulos como *Half-Life* de *Valve* o *Unreal Tournament* de *Epic*, ambos alabados por su excelente IA táctica. *Thief: The Dark Project* de *LGS* destaca por un cuidadoso modelado de la capacidad sensorial de su IA, así como el uso de niveles de alerta graduales que dan al jugador feedbak vital del estado interno de las IAs.

Los juegos de tipo "Sim", tales como *SimCity* de *Maxis*, fueron los primeros en demostrar el potencial de enfoques de vida artificial (A-life). *The Sims*, también de *Maxis*, es un ejemplo del éxito de máquinas de estado borroso (fuzzy-state machine) y tecnologías A-life, que reflejan sus agentes IA. Toda la serie de *Creatures* de *CyberLife* fue otro título que apostó por A-life, convirtiéndose en el mejor ejemplo de aplicación de algoritmos genéticos y redes neuronales artificiales a la industria del entretenimiento. El juego simula con mucho detalle la psicología y fisiología de los "Norns", cada uno de ellos controlado por un "cerebro" virtual implementado por una red de neuronas artificial. Cada criatura recibe información del mundo exterior en el que se desarrolla y su propia fisiología y psicología. La procreación de dos "Norns" produce un huevo, del que nace una nueva criatura, con valores y pesos iniciales determinados por los operadores de cruzamiento y mutación de los cromosomas parentales.

Collin McRae es otro de los videojuegos comerciales donde los NPCs, en este caso conductores oponentes, se controlan usando una red neuronal artificial entrenada mediante un AG. Más que usar el AG para ajustar los pesos de la red, ha habido significantes avances a la hora de aplicar el AG para evolucionar la topología de la red, el número de neuronas y las conexiones entre ellas. Una de las primeras implementaciones usaba cromosomas donde por cada posible conexión en la red (una por cada par de neuronas más una por cada neurona hacia sí misma) un bit indicaba si estaba activa o no (Miller, Todd, & Hedge, 1989). Aproximaciones posteriores buscaron una evolución más genérica de la topología, donde cromosomas codificaban reglas para el crecimiento de la red, más que la red en sí. Un ejemplo interesante es el del algoritmo NEAT (Stanley & Miikkulainen, 2002), que usa información adicional sobre la estructura de la red para evitar que el operador de cruce genere combinaciones desfavorables, que interrumpan el proceso. Existe también una versión de NEAT de tiempo real, rtNEAT, que presenta evaluación y evolución simultánea y asíncrona, usada a modo de experimento en un juego de estrategia en tiempo real sencillo llamado NERO (Stanley, Bryant, & Miikkulainen, 2005). En dicho juego se estableció un entorno configurable de entrenamiento en el que los alumnos podían presentar a los agentes de rtNEAT escenarios de combate cada vez más complejos. Las primeras batallas podían centrarse simplemente en apuntar y atacar al enemigo, mientras que combates avanzados podían incluir terrenos y obstáculos, realizándose todo el entrenamiento en tiempo real.

Los juegos de tipo "Dios" combinan algunos aspectos de los juegos "Sim", como las tecnologías A-life, con elementos de estrategia en tiempo real. Un ejemplo de este tipo sería *Black&White*, de *Lionhead*, que presenta un impresionante componente de aprendizaje, centrado en la Criatura que se debe entrenar a lo largo del juego.

2.3.1 S-ACO y Combat

El videojuego *Combat* se lanzó originalmente por Cleco en 1977 en forma de una consola dedicada. Posteriormente, salió como cartucho en 1982 para Atari 2600. El juego consiste en que dos jugadores controlen un tanque cada uno, con el fin de disparar al vehículo enemigo mientras esquiva los disparos del oponente. Es un escenario clásico muy popular en los juegos de acción online, particularmente los FPS. Los tanques pueden moverse por el escenario, que contiene obstáculos, o pueden romperse y permanecer quietos. También pueden disparar, pero por simplicidad, se considera que pueden hacerlo únicamente en la dirección en la que se están moviendo, sin disponer de torretas giratorias. El escenario se muestra desde un punto de vista aéreo.

A pesar de ser un juego antiguo y relativamente simple, presenta las características básicas de búsqueda de rutas (pathfinding) presentes en todos los juegos modernos de estrategia en tiempo real.

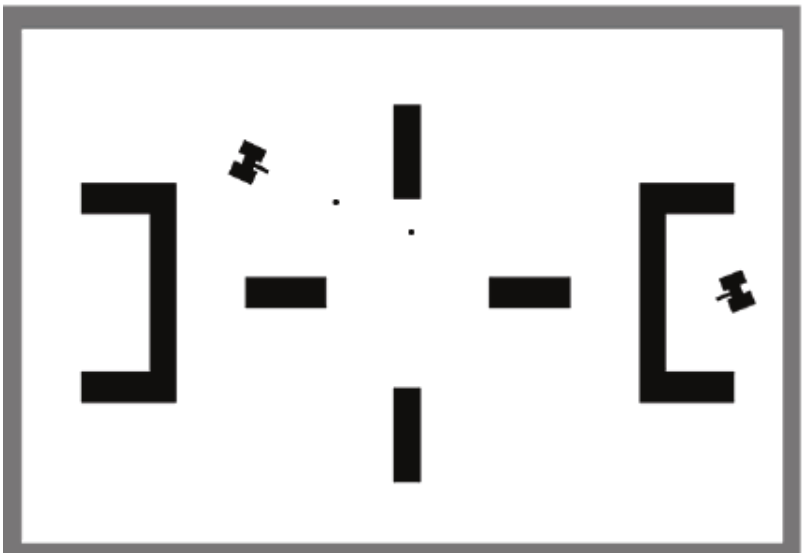


Figura 7: Ejemplo de escenario para Combat, lanzado en 1982 para Atari

Originalmente, era necesario jugar al Combat entre dos jugadores. Las primeras versiones de agentes controlados por IA eran muy crudas y se limitaban a mover el tanque del NPC hacia su objetivo en línea recta. Sin embargo, existen agentes desarrollados aplicando ACO que muestran unos resultados muy convincentes para una versión simplificada del problema. Para el desarrollo del agente, se ha representado el escenario como un conjunto de nodos, formando un grafo como el que muestra la siguiente figura:

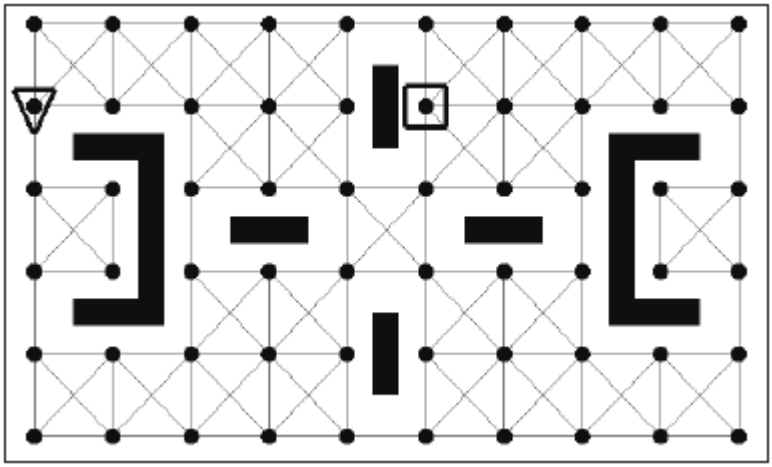


Figura 8: Escenario para Combat modelado para ACO (I)

Los puntos negros representan los nodos del grafo; el triángulo la posición del NPC; el cuadrado la posición del jugador humano y los polígonos negros - los obstáculos que el tanque no puede atravesar.

Los vehículos pueden moverse por todos los nodos, tanto en horizontal, como en vertical o en diagonal. Sin embargo, los pesos de los arcos que unen los nodos no son iguales, ya que los arcos diagonales representan más distancia recorrida que los verticales u horizontales. Por tanto, a un arco entre los nodos *i* y *j* se le asigna un peso de $w_{ij} = \sqrt{2}$ si éste es diagonal y 1 si no lo es. La actualización de feromona es la siguiente:

$$\Delta\tau_{i,j}^{k} = \eta \frac{w_{i,j}}{L^{k}}$$

Fórmula 18: Regla de actualización de feromona para S-ACO en Combat

Las probabilidades de transición se muestran en la siguiente fórmula:

$$p_{i,j}^{k} = \frac{\tau_{i,j}/w_{i,j}}{\sum_{l \in N_j^k} \tau_{il}/w_{il}}, if\ j \in N_i^k$$

Fórmula 19: Regla de transición para S-ACO en Combat

Se aplica este modelo en una simulación del juego, para encontrar la ruta más corta entre el tanque del NPC y el del jugador, que es a su vez un objetivo móvil, teniendo en cuenta que el vehículo no puede atravesar los obstáculos. En el siguiente ejemplo, se muestra la ruta más corta encontrada por el tanque NPC hacia el jugador humano, en una maniobra en la que el jugador humano intenta evadir al oponente:

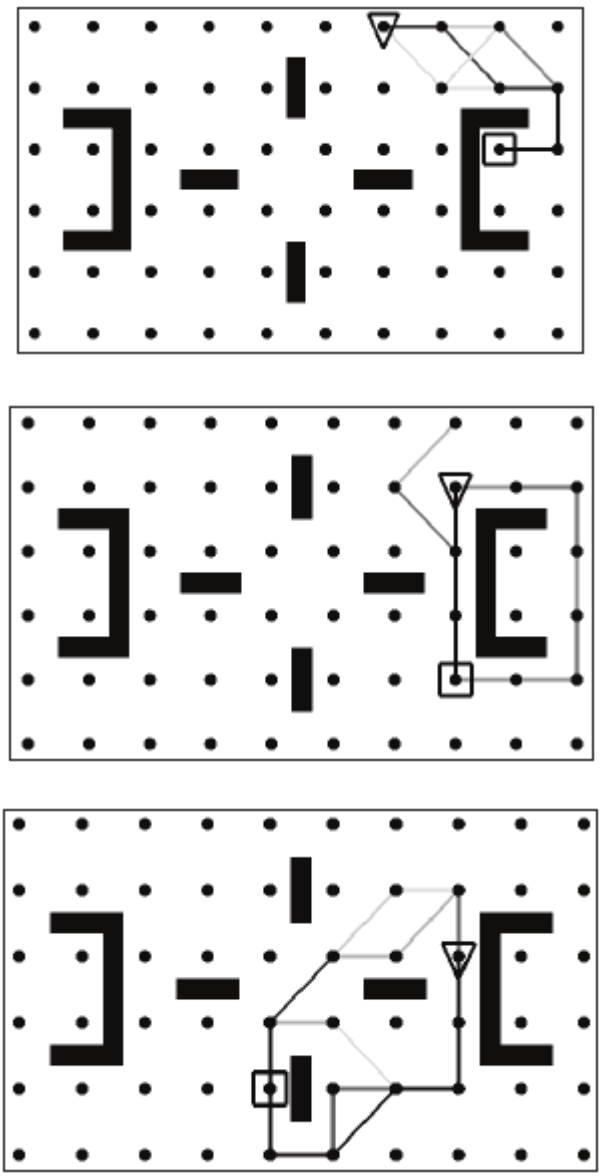


Figura 9: Escenario para Combat modelado para ACO (II)

Donde los arcos más oscuros representan arcos con mayor cantidad de feromona depositada. Una explicación más detallada del planteamiento y la implementación del agente ACO puede verse en Biologically Inspired Artificial Intelligence for Computer Games (Charles, Fyfe, Livingstone, & McGlinchey, 2008)

2.3.2 Pac-mAnt

Pac-Man fue uno de los juegos más populares, desarrollado por Toru Iwatani en 1979 para Namco y rápidamente se convirtió en un éxito mundial. En este juego, el jugador controla a Pac-Man en un laberinto, donde hay dispersas una serie de "píldoras" que Pac-Man debe comer para conseguir puntos. La dificultad aumenta cuando aparecen cuatro fantasmas que intentan dar caza a Pac-Man. Cuando un fantasma y Pac-Man coinciden en una misma posición, se considera que Pac-Man ha sido comido por un fantasma. Sin embargo, existen diferentes tipos de "bolas" esparcidas por el laberinto que

otorgan a Pac-Man puntos y habilidades especiales, como por ejemplo, ser invencible y comerse a los fantasmas.

En 1981 ha salido una nueva versión del juego, llamada Ms. Pac-Man, que comparte las principales características con el juego original. La mayor diferencia radica en el comportamiento de los fantasmas. En Pac-Man, el movimiento de los fantasmas es determinista, lo que significa que si el jugador repite sus movimientos varias veces, los movimientos de los fantasmas también se repetirán. De esta manera, se podía predecir el comportamiento de los fantasmas y ganar la partida sin esfuerzo. Los movimientos de los fantasmas de Ms. Pac-Man incluyen una componente pseudo aleatoria, que complica la predicción su comportamiento e incrementa la dificultad de desarrollar a un agente NPC para el juego.

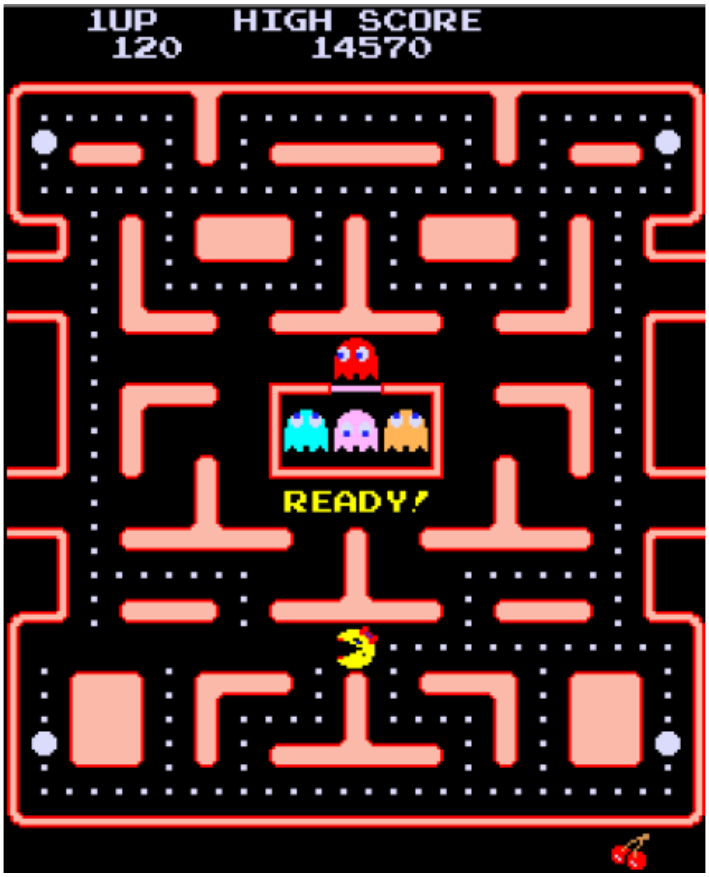


Figura 10: Pantalla del video juego Ms. Pac-Man

En (Martin, Martinez, Recio, & Saez, 2010) se plantea desarrollar un agente basado en algoritmos de Colonias de Hormigas. Puesto que el objetivo es conseguir la mayor puntación posible, el agente debe decidir en todo momento en qué dirección debe realizar su siguiente movimiento basándose en el estado actual de la partida, maximizando la puntuación a la vez que evitando el peligro. Como se ha mencionado más arriba, los puntos se consiguen "comiendo" las "píldoras" esparcidas por el laberinto. Sin embargo, esto no es suficiente para desarrollar a un agente competente, sino que se precisa de técnicas de escape para huir de los fantasmas buscar la ruta más segura para conseguirlo. La combinación de ambas propiedades (agresividad y seguridad) es necesaria para ganar la partida. Por ejemplo, mientras Pac-Man escapa de los fantasmas, si encuentra dos rutas diferentes que contengan "píldoras", deberá escoger la más segura.

Capítulo 3

Planet Wars

En este capítulo se realiza un estudio exhaustivo del problema que se pretende resolver, identificando los distintos subobjetivos a conseguir, con el fin de diseñar y formalizar un modelo que optimice la solución global al problema en su conjunto.

3.1 Análisis del problema

En el presente documento se plantea el desarrollo de un agente para el juego Planet Wars, planteado en el Google AI Challenge 2010, que a su vez fue inspirado en un popular juego para iPhone y ordenador, llamado Galcon, desarrollado por Phil Hassey.

El juego transcurre en un escenario galáctico, donde varios jugadores deben ir adquiriendo tropas espaciales y conquistando planetas. Gana el jugador que consiga reducir las tropas enemigas a 0, o bien obtenga el mayor número de tropas al cabo de un límite de tiempo.

Aunque el juego ofrece la posibilidad de participar a varios jugadores a la vez, el agente se ha desarrollado para una versión simplificada que asume sólo dos jugadores enemigos por partida.

Se comienza a jugar en un mapa compuesto por varios planetas, mostrando simetría central o axial. Cada planeta tiene varias características relevantes para el juego:

- *Ubicación*: Las coordenadas de cada planeta sirven para calcular la distancia relativa de cada planeta con respecto a sus vecinos, medida en turnos que se tardaría en llegar de un planeta a otro.
- *Propietario*: Un planeta puede ser propiedad de uno de los dos jugadores o de ninguno, en cuyo caso se denomina neutral. Todos los planetas son neutrales al comienzo del juego menos dos, los planetas iniciales de los jugadores, que también se colocan simétricamente en el mapa.
- *Coste*: Cada planeta tiene asociado un número de tropas que hace falta enviar para conquistarlo.
- *Tasa de producción*: Cada planeta es capaz de producir un número de tropas cada turno. Un planeta sólo produce tropas si es propiedad de alguno de los jugadores, en cuyo caso las tropas producidas se suman a las ya presentes en el planeta.

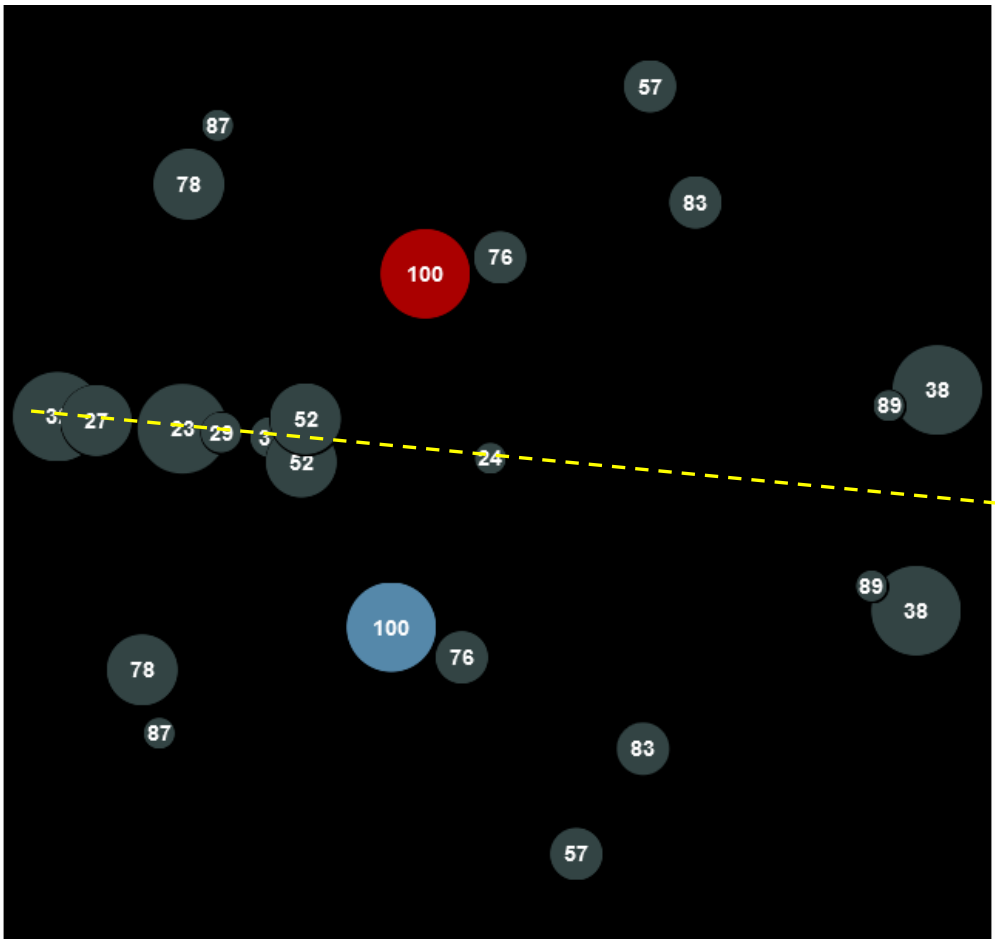


Figura 11: Mapa inicial para una partida de Planet Wars para dos jugadores.

En la figura anterior, se muestra un posible mapa inicial para una partida. Los planetas están simétricos respecto de un eje imaginario, que en este ejemplo se muestra con la línea punteada amarilla. Los planetas neutrales se muestran en gris, cada uno con su conste inicial en tropas. Los planetas de los jugadores se muestran en colores azul y rojo, respectivamente. Cada jugador comienza con 100 tropas y debe usarlas para conquistar más planetas y vencer al enemigo. Éste es un problema complejo, donde el agente deberá considerar múltiples factores en cada instante del tiempo, para tomar la mejor decisión en cada momento basándose en el estado actual de la partida. Por ello, el problema general se ha dividido en varios subproblemas más sencillos, simplificando así

el diseño del agente propuesto, y desarrollando iterativamente versiones sucesivas y más capaces del mismo.

- **Expansión**: El agente, al igual que los demás jugadores, comienza el juego con 100 tropas en un único planeta. Atacar de lleno al planeta enemigo no suele ser una acción que lleve al éxito, puesto que ambos planetas tienen el mismo factor de crecimiento (GR – growth rate), producen la misma cantidad de tropas por turno, y la distancia entre ellos juega a favor del jugador atacado si éste no gasta tropas. Por ejemplo, en el escenario descrito en la Figura 11, la máxima cantidad de tropas que puede mandarle un jugador a otro en el turno 0 es 100, que es la misma cantidad que tienen los dos jugadores por ser el número de tropas inicial. Si la distancia entre ambos planetas iniciales es de t turnos, y el agente lanza un ataque de 100 tropas, cuando esas tropas lleguen, el planeta enemigo ya no tendrá únicamente las 100 tropas iniciales, sino que habrá producido además GR tropas cada turno. Por lo tanto, el enemigo tendrá 100+t*GR. Al restarle las 100 tropas que pierde en combate, el enemigo se quedará con t*GR tropas. El agente, a su vez, se quedará con 0 tropas en el planeta origen, el cual producirá las mismas GR tropas cada turno. Por lo tanto, al cabo de t turnos, cuando el ataque del agente llegue a su destino, ambos planetas tendrán t*GR tropas, volviendo al escenario inicial en el que ambos jugadores tienen un planeta cada uno, con el mismo GR y el mismo número de tropas iniciales en ambos.

  Una vez visto que el ataque inicial al planeta enemigo no es un movimiento ganador, el primer problema que se le plantea al agente es la expansión. El agente debe decir cuántas tropas y desde qué planetas debe mandar a qué planetas destinos para maximizar su producción de tropas y tener cantidad suficiente como para vencer.
- **Defensa**. El enemigo también está jugando la partida, por lo que, inevitablemente, llegará un momento en el juego en el que ambos querrán luchar por el mismo recurso. Cuando el oponente quiera expandirse conquistando alguno de los planetas del agente, éste deberá decidir cómo de ventajoso es mandar tropas suyas al planeta para evitar su pérdida, y desde qué planetas y en qué orden hacerlo. Además, también deberá decidir si la defensa deberá primar sobre la conquista de algún otro planeta, ya que los recursos (tropas) de los que dispone son limitadas y no siempre puede hacer las dos cosas.

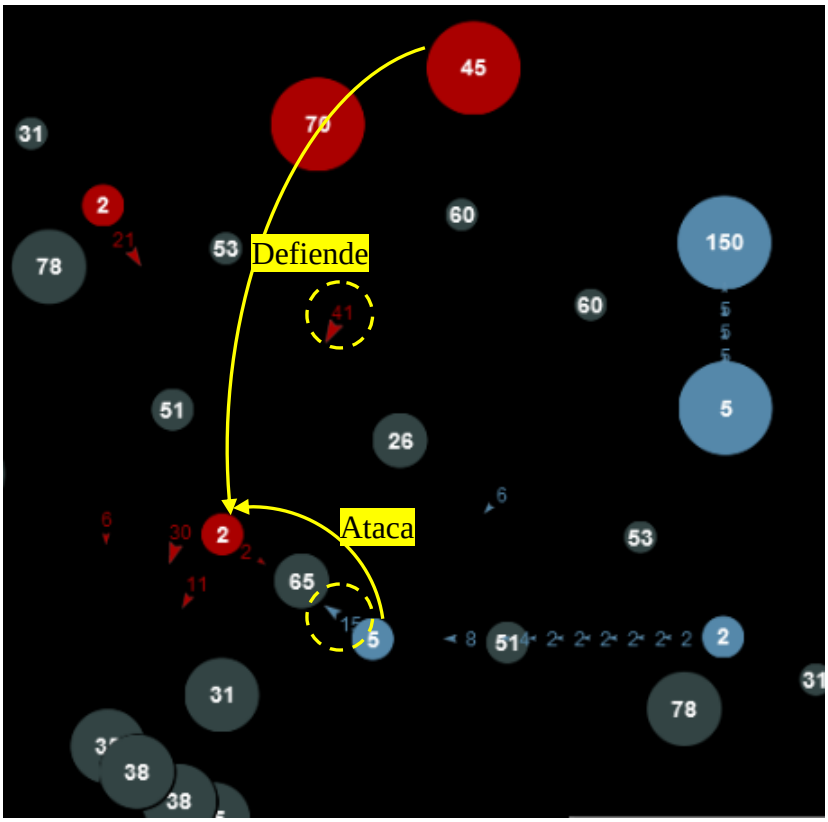


Figura 12: Defensa de un planeta en Planet Wars

- **Balanceo de tropas.** Por último, puede darse el caso de que, tras un turno, le sobren tropas al agente que no hayan sido dedicadas a la conquista o a la defensa. El agente deberá decidir qué hacer con dichas tropas y dónde colocarlas, para que le aporten la mayor ventaja estratégica en el futuro.

3.2 Modelado

3.2.1 Expansión

Para modelar este problema, el mapa del juego se representa como un grafo, en el que los nodos se corresponden con planetas, cada uno con la información correspondiente del propietario, posición, tropas disponibles, GR, etc.

Las aristas del grafo unen los planetas y representan las posibles rutas de las tropas que el agente puede mandar. Éste puede mandar tropas desde cualquier planeta del que es propietario a cualquier otro, ya sea suyo o no. Por lo tanto, un planeta del agente estará totalmente conectado con todos los planetas del grafo. Cada arco del grafo contendrá información sobre su longitud (D), coste (C) y, posteriormente, la heurística (η) y la feromona (τ) depositada en el mismo.

Sean $P = \{P_0,...,P_n\}$ el conjunto de planetas de la partida y $P_{jug} = \{P_x,...,P_y\}$ el conjunto de planetas conquistados por el jugador *jug*. La información del grafo se

representa mediante una matriz bidimensional A de nxn posiciones, donde $a_{ij} \in A$ indica la información sobre el arco que une el planeta i con el planeta j en el sentido desde i a j:

$$A = \begin{pmatrix} a11, a12, \ldots, a1n \\ a21, a22, \ldots, a2n \\ \ldots \\ an1, an2, \ldots, ann \end{pmatrix}$$

Fórmula 20: Estructura de almacenamiento de información sobre el grafo de Planet Wars

El agente también debe cumplir ciertas restricciones. Primero, un planeta no puede mandarse tropas a sí mismo. Esta acción no tendría sentido, puesto que la distancia de un planeta a sí mismo es 0 y las tropas llegarían instantáneamente. El mismo resultado se consigue no realizando ninguna acción y simplemente dejando pasar un turno acumulando tropas.

En segundo lugar, el agente no puede mandar tropas desde planetas que no sean suyos. Así, mandar 10 tropas desde el planeta i, que es del agente, al planeta j, que no es suyo, sería una acción permisible, mientras que hacerlo en sentido contrario no lo sería. Esto demuestra que diferentes arcos tendrán costes diferentes e, incluso, el mismo arco tendrá costes distintos dependiendo del sentido en el que se recorra.

En este punto, el conjunto de restricciones para el problema sería el siguiente:

$$aij \neq aji,$$
$$aii = \infty$$
$$aij = \infty \text{ si } i \notin P_{jug}.$$

Fórmula 21: Conjunto restricciones (I) para agente de Planet Wars.

Una vez definidas las estructuras de datos para representar el escenario del juego, se procede a describir el funcionamiento del modelo de hormigas que emplea el agente. El objetivo de dicho modelo es construir una solución que indique qué planetas, y en qué orden, deben conquistarse para maximizar el beneficio, cumpliendo las restricciones del problema. Normalmente, una hormiga comenzará a construir una solución partiendo de su planeta inicio, que debe ser obligatoriamente un planeta del agente. Una vez que el agente haya conquistado más planetas, habrá hormigas que salgan de cada una de ellas. A continuación, cada hormiga deberá evaluar los posibles planetas a los que puede moverse y realizar el salto. El conjunto de los planetas candidatos estará formado por todos los planetas que no sean del agente y que no hayan sido visitados antes, para evitar los bucles. Sean P_{cand} y P_{visit} los conjuntos de planetas candidatos y visitados, respectivamente:

$$P_{visit} = \{P_a, \ldots, P_m\}$$

$$P_{cand} = \{P_n, \ldots, P_z\} \text{ donde } P_i \notin P_{jug} \forall \text{ i}$$
$$P_i \notin P_{visit} \forall \text{ i}$$

Fórmula 22: Definición de conjuntos de planetas visitados y candidatos

Por lo tanto, en cada iteración, cada hormiga evaluará P_{cand} y P_{visit} y calculará el siguiente planeta destino usando la siguiente regla de transición:

$$T_{ij} \begin{cases} max(\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}) \text{ si } Q < Q_0 \\ J \text{ si } Q >= Q_0 \end{cases}$$

Fórmula 23: Regla de transición para Planet Wars

Donde Q_0 es un parámetro tal que $0<Q_0<1$, que determina la importancia relativa entre la explotación y la exploración; J es una variable aleatoria con función de probabilidades dada por la ecuación descrita en la Fórmula 8.

En este punto, el objetivo del agente es aumentar sus tropas de tal manera que tenga potencia de fuego suficiente para superar al enemigo. La única manera de aumentar las tropas de un jugador es consiguiendo planetas que, una vez conquistados, produzcan tropas cada turno. Sin embargo, para conquistar un planeta es necesario sacrificar tropas. Por lo tanto, es necesario tener en cuenta tanto la productividad del planeta, también llamado Growth Rate (GR), como su coste en tropas. Además, la presencia de un contrincante que puede responder y anticiparse a los movimientos del agente y la limitación del combate en el tiempo, hace que este último sea otro factor clave a la hora de elegir qué planeta conquistar. En otras palabras, la función que determinará cómo de ventajoso será conquistar un planeta en particular, y no otro, debe basarse en al menos una, si no todas, de estas tres variables. Y dicha función será la función heurística η que indique la visibilidad de cada nodo.

Se han realizado pruebas con diferentes heurísticas en las primeras fases del desarrollo del agente:

(1) $\eta=1/D$	– Elige primero el planeta más cercano
(2) $\eta=1/C$	– Elige primero el planeta más barato
(3) $\eta=GR/C$	– Elige primero el planeta con mejor ratio de crecimiento/coste
(4) $\eta=GR/(D^2*C)$	– Elige primero el planeta con mejor ratio de crecimiento/(distancia y coste)

Figura 13: Funciones heurísticas para la expansión del agente.

La descripción y el detalle esas pruebas se presentan en la sección 4.1 Evaluación de las heurísticas planteadas. Finalmente la heurística que obtuvo mejores resultados fue la (4), por lo que se mantuvo en la versión final del agente.

A medida que se construye la solución, las hormigas recorren distintos arcos y actualizan la feromona de éstos, favoreciendo que las demás hormigas recorran caminos diferentes. La actualización local se realiza inmediatamente después de que una hormiga haya transitado por un arco, según la ecuación de la Fórmula 24:

$$\tau_{ij}^{t+1}=(1-\sigma)*\tau_{ij}^{t}+\sigma*\tau_{ij}^{0}$$
$$0<\sigma<1$$
$$\tau_{ij}^{0}=1/(n*Lmin)$$

Fórmula 24: Regla de evaporación de feromona para Planet Wars

Donde $\tau_{i,j}$ representa la feromona para el arco *(i,j)*, σ representa el decaimiento del nivel de feromona, $\tau^{0}_{i,j}$ es el valor inicial de feromona y *Lmin* es el camino más corto que pasa por todos los planetas, midiendo la distancia en turnos.

Una vez que la hormiga realiza el salto, el planeta destino del mismo se traspasa del conjunto de planetas candidatos P_{cand} al conjunto de planetas visitados P_{visit}. Cuando el conjunto de planetas candidatos quede vacío, la hormiga ya no tendrá la posibilidad de realizar más saltos, por lo que finalizará su turno.

Cuando todas las hormigas hayan acabado sus movimientos, se evaluarán las soluciones construidas por cada una de ellas. Para definir los criterios que permitan comparar dos soluciones entre sí, debe considerarse el objetivo del subproblema que se está planteando en este apartado. Como se ha mencionado anteriormente, el fin último del agente en esta etapa es la generación de tropas, a través de la expansión. Por ello, es lógico que el principal criterio de comparación sea el número de tropas generadas. Para ajustar aún más este indicador, se consideran como tropas generadas únicamente las tropas ganadas, habiéndoles restado las tropas gastadas para la conquista de cada uno de los planetas. Además, se impone una limitación temporal de *t* turnos, que es la duración de un combate estándar. De este modo, la función de fitness que usará el agente para determinar la calidad de cada solución se ha implementado como sigue:

$$F = \Sigma \ ((t\text{-}D(i,k))*GR(Pk)\text{-}C(Pk))$$
$$\forall \ Pk \in P_{jug}$$
$$\#P_{jug} = n$$

Fórmula 25: Función de Fitness en Planet Wars

Donde t es el límite de turnos del combate; $GR(P_k)$ y $C(P_k)$ son el factor de crecimiento de tropas y el coste de cada planeta, respectivamente; $D(i,k)$ es la distancia, en turnos, desde el planeta inicial P_i al planeta conquistado P_k; y P_{jug} es el conjunto de planetas conquistados por el jugador para la solución evaluada, siendo *n* su cardinalidad.

Por ejemplo, sea el siguiente el escenario de batalla en curso:

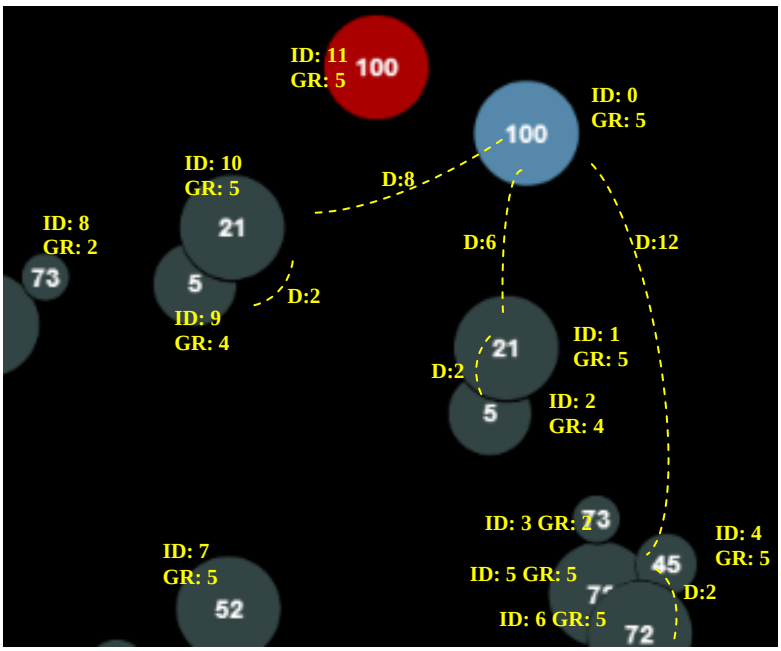


Figura 14: Ejemplo mapa Planet Wars

Para seguir mejor el ejemplo, cada planeta aparece en el mapa con la información de su ID, Growth Rate y Distancia. Sea 0 el planeta del agente (marcado en azul) y 11 el planeta del contrincante (marcado en rojo). La duración del combate se establece en 100 turnos. Supongamos que las hormigas del agente han encontrado las siguientes soluciones, tras finalizar el primer turno:

Solución 1: Conquistar planetas 1 y 2.
En este caso, la función fitness será:

$$\mathcal{F} = \Sigma_{k=1}^{2} ((t\text{-}D(0,k))*GR(Pk)\text{-}C(Pk))$$
$$\#\, P_{jug} = 2$$
$$P_{jug} = \{1,2\}$$

Sustituyendo las variables con los valores del ejemplo:

$$\mathcal{F}\,(P_1) = ((100\text{-}6)*5\text{-}21) = 449$$
$$\mathcal{F}\,(P_2) = ((100\text{-}8)*4\text{-}5) = 363$$
$$\mathcal{F}_{total} = \Delta\tau(P_1) + \Delta\tau(P_2) = 812$$

Solución 2: Conquistar planetas 10 y 9.
El valor de fitness de esta solución será:

$$\mathcal{F}\,(P_{10}) = ((100\text{-}8)*5\text{-}21) = 439$$
$$\mathcal{F}\,(P_9) = ((100\text{-}10)*4\text{-}5) = 355$$
$$\mathcal{F}_{total} = \Delta\tau(P_1) + \Delta\tau(P_2) = 794$$

Solución 3: Conquistar planetas 5 y 6.
El fitness en este caso será:

$$\mathcal{F}\,(P_5) = ((100\text{-}12)*5\text{-}73) = 367$$
$$\mathcal{F}\,(P_6) = ((100\text{-}14)*5\text{-}72) = 358$$
$$\mathcal{F}_{total} = \Delta\tau(P_1) + \Delta\tau(P_2) = 725$$

El agente considerará como mejor solución aquella que maximice el valor de fitness. En el ejemplo mostrado, la solución de mejor calidad es la primera, que pasa por conquistar los planetas 1 y 2.

Una vez evaluadas todas las soluciones, se actualiza la feromona usando la mejor solución candidata encontrada, según muestra la Fórmula 26:

$$\tau_{ij}^{t+1} = \rho * \tau_{ij}^{t} + (1-\rho) * \Delta\tau_{ij}$$
$$0 < \rho < 1$$
$$\rho = 1-\sigma$$

Fórmula 26: Regla de actualización global de feromona para Planet Wars.

Donde $\tau_{i,j}$ es el valor de feromona para el camino entre planetas P_i, y P_j; ρ es el factor de actualización de feromona y $\Delta\tau_{i,j}$ es el aporte, en tropas, de la mejor solución global encontrada hasta el momento si el componente *i,j* forma parte de dicha solución o *0* en caso contrario.

3.2.2 Defensa

Este problema es muy diferente al anterior y se plantea al agente por el mero hecho de tener que enfrentarse a un oponente. El bando opuesto de la batalla tiene el mismo objetivo y los mismos recursos que el agente, por lo que también intentará expandirse a través de la conquista de planetas. En teoría, llegará un momento cuando no queden planetas neutrales, por lo que la expansión de un bando pasará por quitarle planetas al bando contrario. En la práctica, este momento llega mucho antes, ya que las mejores rutas de expansión pasan por planetas ya conquistados, a pesar de existir planetas neutrales disponibles que, sin embargo, no forman parte de las soluciones con más probabilidades de ganar. Además, no hay que olvidar que la expansión no es el fin en sí, sino que es sólo un medio hacia el objetivo final, que es ganar al oponente, bien quitándole todas sus tropas, bien acumulando más tropas que él al final del combate. Por eso, en toda batalla llega un momento en el que un jugador considera que tiene suficientes tropas como para dejar de interesarle conquistar planetas neutrales y comenzar a atacar planetas del contrincante. Ese momento puede llegar antes o después, dependiendo de si un jugador o su enemigo siguen una estrategia más o menos agresiva.

Este problema se ha modelado de una manera muy parecida al de la conquista, ya que comparten múltiples características, relativas sobre todo a la representación del entorno del combate. No obstante hay diferencias fundamentales derivadas del contraste entre los objetivos de cada subproblema. En el caso de la expansión, la meta del agente es la elección de los planetas más adecuados para la generación de tropas. En el caso de la defensa, el principal objetivo del agente será la elección de los planetas amenazados por el enemigo que deban ser protegidos para evitar su caída en manos del oponente. Debe tenerse en cuenta que proteger un planeta implica enviarle tropas que, de otra maneta, podrían usarse para conquistar otros planetas o atacar al enemigo. Por lo tanto, es importante que el agente evalúe acertadamente qué planetas propios son lo suficientemente importantes como para gastar recursos en su preservación y qué planetas pueden sacrificarse al contrincante.

Antes de entrar en los detalles de cómo se evalúan todos esos factores, se explicará brevemente la representación de este problema. Al igual que en el apartado anterior, el mapa del juego se representa como un grafo, donde los nodos se corresponden con planetas. Sean $P = \{P_0,\ldots,P_n\}$ el conjunto de planetas de la partida y $P_{jug} = \{P_x,\ldots,P_y\}$ el conjunto de planetas conquistados por el jugador *jug*. La información de los mismos se recoge en la estructura mostrada en la Fórmula 20.

El agente debe cumplir una serie de restricciones en este caso también, aunque ahora se introducen variaciones con respecto al escenario anterior. La primera restricción mantiene la prohibición de mandar tropas desde un planeta a sí mismo y no varía con respecto al problema de la conquista. La segunda restricción tampoco cambia, impidiendo mandar tropas desde un planeta que no pertenezca al jugador en cuestión. La tercera condición es la que presenta modificaciones, ya que en el caso de la defensa, no tiene sentido que el agente proteja un planeta si éste no le pertenece. En el caso de un planeta enemigo o neutro, mandarle tropas se correspondería con un caso de conquista, ya sea con contienda de por medio o no. Sin embargo, no se puede defender algo que aún no se posee. Por lo tanto, la tercera y última restricción, en ese caso, sería que no es permisible mandar tropas a un planeta no propio.

Resumiendo estos últimos puntos, el conjunto de restricciones para el problema de defensa sería el siguiente:

$$aij \neq aji$$
$$aii = \infty$$
$$aij = \infty \text{ si } i \notin P_{jug}$$
$$aij = \infty \text{ si } j \notin P_{jug}$$

Fórmula 27: Conjunto restricciones (II) para agente de Planet Wars.

Como ya se ha mencionado, el objetivo del agente en esta fase es construir una solución que indique qué planetas y en qué orden, deben defenderse para conservar el máximo beneficio en tropas sobre el enemigo, siempre cumpliendo las restricciones impuestas. Una hormiga comenzará a construir una solución desde el planeta de inicio, que debe ser un planeta propio del agente. Una vez que el agente haya conquistado varios planetas, habrá hormigas que salgan de cada uno de ellos, ya que todos los planetas pueden participar en la defensa. Así, partiendo de un planeta origen, cada hormiga deberá evaluar los posibles planetas a los que puede moverse, antes de realizar el salto. El conjunto de los planetas candidatos estará formado por todos los planetas que sean del agente y que estén siendo atacados. Para determinar esta última condición, se estudian todas las tropas que haya en vuelo en el momento actual de la batalla. Se considera que un planeta P_j está siendo atacado si hay alguna tropa *Fl(i,j)* del enemigo cuyo planeta destino sea P_j.

Sean P_{cand} y P_{visit} los conjuntos de planetas candidatos y visitados, respectivamente. Sean F_{jug} y F_{ene} los conjuntos de tropas del jugador y del enemigo, respectivamente:

$$P_{visit} = \{P_k,\ldots,P_n\}$$

$$P_{cand} = \{P_x,\ldots,P_z\} \text{ donde } P_i \in P_{jug} \forall \text{ i}$$
$$P_i \notin P_{visit} \forall \text{ i}$$
$$\exists Fl(j,i) \in Fl_{ene} \forall \text{ i}$$

Fórmula 28: Definición de conjuntos de planetas visitados y candidatos (II)

En cada iteración, la hormiga evaluará P_{cand} y P_{visit} y calculará el siguiente planeta destino usando la regla de transición descrita en la Fórmula 23.

A la hora de calcular las probabilidades de transición entra en juego el factor de la visibilidad del nodo. Es decir, cómo de atractivo o beneficioso es este nodo, en función de los objetivos que se persiguen. En el caso de la conquista, la visibilidad o heurística η estaba estrechamente relacionada con la productividad del planeta destino (GR), la distancia a la que se encuentra y su coste en tropas. En el caso de la defensa, la situación es muy similar. En primer lugar, el Growth Rate del planeta sigue siendo muy importante ya que indica la cantidad de tropas que ese planeta es capaz de generar por turno. Esta medida puede ser igualmente representativa del interés que puede suponer tanto para el enemigo a la hora de atacarlo como para el agente a la hora de conservarlo. Si un planeta genera muchas tropas, puede ser atacado con mayor probabilidad, sobre todo si en algún momento el agente se despista y lo deja desprotegido.

En segundo lugar, la distancia también sigue siendo un factor de peso, ya que puede afectar al interés que pueda tener el agente en defender un planeta destino desde un planeta origen dado. El principal inconveniente en mandar las tropas a un planeta lejano, ya sea para la conquista como para la defensa, es que uno no puede disponer de las tropas en vuelo. Una vez mandadas, no se puede cambiar el curso de las tropas ni volverlas atrás. Es decir, no queda más remedio que esperar a que las tropas lleguen a su destino

antes de poder mandarlas a otro lugar. Esto comienza a ser un problema en trayectos largos, ya que la situación de la batalla cambia continuamente y es posible que una acción que parecía rentable en el tuno 0 ya no lo sea en el turno 10. Si las tropas se han mandado a un planeta que está a 20 turnos de distancia, el agente no puede reaccionar al cambio de la situación hasta 10 turnos después de que se haya producido. Por ello, estando en un planeta determinado, defender otro planeta que está muy lejos puede que sea muy arriesgado, ya que a mayor distancia, menor capacidad de reacción del agente. Seguramente, será más rentable dedicar las tropas de ese planeta origen a la conquista, y defender el planeta lejano usando tropas de otros planetas más cercanos a él.

Por último, queda el factor del coste. El coste en la conquista representaba la inversión en tropas que el agente debía hacer para conseguir los beneficios de un recurso. Básicamente, representaba el número de tropas existentes en un planeta neutral o enemigo. Sin embargo, esta descripción de coste no es aplicable a este subproblema, ya que, por la propia definición que se ha hecho de la defensa, sólo se pueden defender los planetas que ya se poseen. En consecuencia, no existe ningún número de tropas que se debe superar ya que todas las tropas de ese planeta son recursos propios. Por lo tanto, se sustituye el coste por otro factor, llamado Vulnerabilidad (*V*). En el marco de Planet Wars se concibe el peligro como la diferencia entre las tropas enemigas dirigidas a un planeta y las tropas propias existentes en ese planeta en el momento del ataque:

$$V(i) = Fl(i) + \sum_{t=0}^{k} (GR(i) + \Sigma Fl^{t}(j,i) - \Sigma Fl'^{t}(j',i))$$

$Fl^{t}(j,i) \in Fl_{jug} \; \forall$ j
$Fl'^{t}(j,i) \in Fl_{ene} \; \forall$ j´

Fórmula 29: Vulnerabilidad de un planeta en Planet Wars.

Donde *V(i)* es la vulnerabilidad del planeta *i*; *Fl(i)* son las tropas del planeta *i*; *GR(i)* es la tasa de productividad del planeta *i*; $\Sigma Fl^{t}(j,i)$ es la suma de todas las tropas amigas que se dirigen al planeta *i* y llegarán en el turno *t*; $\Sigma Fl'^{t}(j',i)$ es la suma de todas las tropas enemigas que se dirigen al planeta *i* y llegarán en el turno *t*. Nótese que el cálculo se realiza en un marco acotado de tiempo de *k* turnos.

Evidentemente, cuanto mayor es el peligro al que se enfrenta un planeta, mayor es la necesidad de defenderlo.

Habiendo estudiado los diferentes factores que pueden afectar, a priori, a la necesidad de proteger un planeta y otro, la función heurística para la defensa queda como sigue:

$$(1)\ \eta = (GR * V)/(1+D)$$ – *Elige primero el planeta con mejor ratio de crecimiento y vulnerabilidad/distancia*

Fórmula 30: Función heurística para la defensa del agente.

Las reglas de actualización de feromona local y global son iguales a las indicadas en las Fórmulas 24 y 26, respectivamente, con la diferencia de actuar sobre otros valores de feromona diferentes, para no afectar al modelo expansivo. La feromona defensiva se representará como τ^{d}_{ij}:

$$\tau^{d}_{ij}{}^{t+1} = (1-\sigma) * \tau^{d}_{ij}{}^{t} + \sigma * \tau^{d}_{ij}{}^{0}$$
$$0<\sigma<1$$
$$\tau^{d}_{ij}{}^{0} = 1/(n*Lmin)$$

Fórmula 31: Regla de evaporación de feromona para defensa Planet Wars

$$\tau_{ij}{}^{t+1} = \rho * \tau_{ij}{}^{t} + (1-\rho) * \Delta\, \tau_{ij}$$
$$0<\rho<1$$
$$\rho = 1-\sigma$$

Fórmula 32: Regla de actualización global de feromona para defensa Planet Wars

Para poder aumentar la feromona de la mejor solución cada turno, se debe disponer de una medida de comparación entre dos o más soluciones que permita determinar cuál prima sobre las demás. Con este objetivo en mente, se puede ver la protección de un planeta como una acción equivalente a la conquista de ese mismo planeta que, de no hacer nada, pasaría a manos enemigas. En otras palabras, defender un planeta garantiza una serie de recursos que no se obtendrían de otra manera. Partiendo de esta base, se puede aplicar la misma función de fitness que en el subproblema de la expansión (ver Fórmula 25), considerando como mejor solución aquella que más tropas permita ganar (conservar en este caso) en un período acotado de tiempo.

3.2.3 Balanceo de Tropas

En las dos secciones anteriores se han descrito dos componentes muy importantes de la batalla: el ataque y la defensa. Si bien uno puede pensar que ambos en conjunto pueden asegurar el éxito de un combarte, queda otro componente que puede ser decisivo a la hora de proclamarse vencedor.

Como ya se ha descrito en la sección anterior, la imposibilidad de cambiar la trayectoria de las flotas una vez mandadas hace que sea poco rentable mandar tropas en trayectos de larga distancia, ya que reduce el número de tropas disponibles para reaccionar ante un cambio de situación en el combate, que puede provocarse debido a acciones del enemigo que el agente no controla. Por ello, en términos de velocidad de reacción y capacidad de contraataque, resulta más ventajoso resolver los conflictos de forma local. Es decir, para conquistar o defender un planeta es mejor usar tropas localizadas en los planetas más cercanos que otras que estén más lejos. Esto desemboca en la necesidad de tener siempre tropas disponibles en ubicaciones cercanas a los puntos calientes del combate. Esta necesidad se ha denominado Balanceo de Tropas.

La definición que se ha dado del Balanceo de Tropas plantea dos cuestiones: ¿Cuáles son los puntos calientes del combate? y ¿Qué tropas y en qué momento se acumulan en esos puntos?

Puntos calientes

Este concepto puede explicarse más claramente si se compara el combate de Planet Wars con una campaña militar en la vida real. En este caso, cuando dos enemigos ocupan cierta extensión del terreno, una junto a la otra, lo normal es que las batallas no se produzcan a lo largo de todo el mapa por igual. Lo habitual es que haya más encuentros en la zona más cercana al enemigo, comúnmente llamada el “frente”. Sin embargo, cuanto más alejado está un punto de la frontera enemiga, más seguro se encuentra, ya que

está protegido por tropas amigas. Por lo tanto, en estos puntos, que se llamarán la retaguardia, las probabilidades de batalla son menores.

Esta misma situación se produce en Planet Wars. En la siguiente figura se muestra un estado avanzado de una batalla:

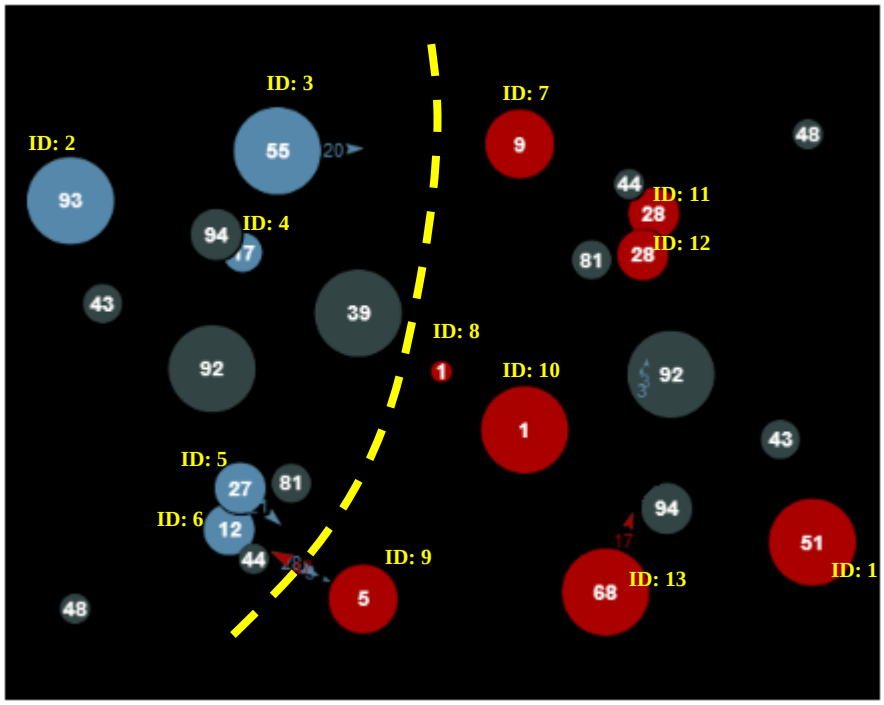


Figura 15: Ejemplo batalla Planet Wars.

Se puede intuir, sin necesidad de hacer cálculos, que el mapa está dividido aproximadamente por igual entre los dos jugadores, llevando el jugador rojo cierta ventaja. Asimismo, se puede apreciar una frontera imaginaria, que en la imagen está marcada con una línea discontinua amarilla y que marca una frontera invisible entre las fuerzas enfrentadas, es decir, la línea del frente.

Los planetas 1 y 2 están muy alejados de la línea del frente. Si el oponente azul quisiera conquistar, por ejemplo, el planeta 1, necesitaría mandar un número considerable de tropas por dos razones. La primera es el GR del planeta 1. Como éste está muy retirado, las tropas tardarán muchos turnos en llegar, tiempo durante el cual el planeta seguirá acumulando tropas a favor de su dueño. La segunda razón es que el jugador azul, atacando desde el otro lado del frente, estará siempre más lejos del planeta 1 que los planetas del jugador rojo. Eso le da ventaja al defensor, ya que tiene más tiempo para reaccionar al movimiento enemigo ya que siempre dispone de planetas que están más cerca del atacado, por lo que los refuerzos llegarán antes que el ataque. Debido a eso, resulta muy costoso atacar un planeta que se encuentra muy adentrado en terreno del contrincante y, por eso, la retaguardia es una zona más segura.

Sin embargo, no sucede lo mismo con planetas cercanos a la línea del frente. En esta zona, los planetas aliados y enemigos están cerca los unos de los otros y ambos bandos cuentan con un refuerzo que puede venir desde su retaguardia. Por lo tanto, las probabilidades de ganar un encuentro son mayores y eso causa que ambos oponentes sean más atrevidos a la hora de provocar escaramuzas, aumentando las posibilidades de que surja combate.

Para hallar las áreas del mapa de batalla que cumplan las condiciones de zona caliente, se ha implantado un índice de peligrosidad (*IP*) para cada planeta. Este índice se

define como el número de planetas enemigos de los que un planeta aliado está más cerca que ningún otro planeta aliado:

$$IP_p = \sum_{i=0}^{i=k} 1 \quad \forall\, i \in P_{enemigo}$$
$$Tal\,que\;\; D(p,i) < D(j,i) \;\; \forall\, i \in P_{jug}$$

Fórmula 33: Índice de Peligrosidad en Planet Wars.

Según esta definición, el índice de peligrosidad será tanto más alto cuanto más cerca esté un planeta de los planetas enemigos en relación con otros planetas aliados, indicando de este modo aquellos que estén más adentrados en la zona fronteriza.

Partiendo del mapa de la Figura 15, se presenta un ejemplo del cálculo del IP para algunos planetas:

Ejemplo 1: IP del planeta 3

Se considera la distancia del planeta 3 a cada uno de los planetas enemigos: 7, 8, 9, 10, 11, 12 y13. Se le suma un 1 al índice de peligrosidad del planeta 3 por cada planeta rojo que esté más cercano a él que al resto de los planetas azules. Por lo tanto:

El planeta 3 es el planeta azul más cercano al 7. Se suma 1 punto al IP_3.

El planeta 3 NO es el planeta azul más cercano al 8. La distancia entre el planeta 4 y el planeta 8 es la más corta.

El planeta 3 NO es el planeta azul más cercano al 9. La distancia entre el planeta 6 y el planeta 8 es la más corta.

El planeta 3 NO es el planeta azul más cercano al 10. La distancia entre el planeta 5 y el planeta 10 es la más corta.

El planeta 3 es el planeta azul más cercano al 11. Se suma 1 punto al IP_3.

El planeta 3 es el planeta azul más cercano al 12. Se suma 1 punto al IP_3.

El planeta 3 NO es el planeta azul más cercano al 13. La distancia entre el planeta 5 (y el 6) y el planeta 13 es la más corta.

El planeta 3 NO es el planeta azul más cercano al 1. La distancia entre el planeta 5 y el planeta 1 es la más corta.

Tras finalizar estos cálculos, el IP_3 habrá sumado 3 puntos.

Ejemplo 2: IP del planeta 2

Se considera la distancia del planeta 2 al resto de los planetas enemigos: 7, 8, 9, 10, 11, 12 y13. El IP_2 quedará así:

El planeta 2 NO es el planeta azul más cercano al 7. La distancia entre el planeta 3 y el planeta 7 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 8. La distancia entre el planeta 4 y el planeta 8 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 9. La distancia entre el planeta 6 y el planeta 8 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 10. La distancia entre el planeta 5 y el planeta 10 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 11. La distancia entre el planeta 3 y el planeta 11 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 11. La distancia entre el planeta 3 y el planeta 11 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 13. La distancia entre el planeta 5 (y el 6) y el planeta 13 es la más corta.

El planeta 2 NO es el planeta azul más cercano al 1. La distancia entre el planeta 5 y el planeta 1 es la más corta.

Tras finalizar estos cálculos, el IP_2 habrá sumado 0 puntos.

Una vez calculado el IP de todos los planetas conquistados del mapa, el resultado se convierte, a su vez, en otro mapa que indica la proximidad de un punto a la frontera imaginaria entre los dos ejércitos. Si se aplicaran diferentes colores a los diferentes IP, el escenario de batalla se convertiría en un mapa de "calor", tal y como muestra la siguiente figura:

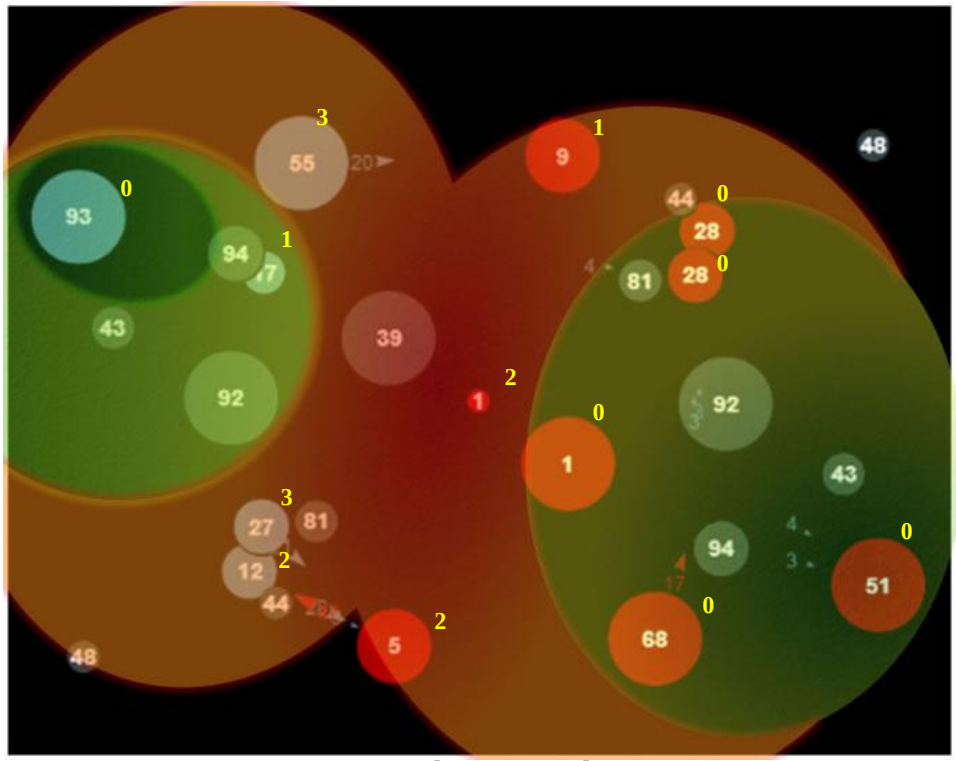


Figura 16: Mapa de IP para Planet Wars

Una vez determinadas las distintas regiones del mapa, la siguiente acción consiste en pasar las tropas desde las áreas de menor riesgo hacia aquellas donde el peligro es mayor. Este traspaso se realiza después de haber decidido todos los movimientos de ataque y defensa. En ese momento, todas las tropas sobrantes que se hayan generado en los planetas conquistados y no se hayan usado, se mandarán al planeta más próximo con mayor índice de peligrosidad. De esta manera se asegura el movimiento del exceso de tropas que se genera en la retaguardia y que es difícil de rentabilizar debido a su ubicación y se posicionan en planetas más activos, donde van a emplearse para futuras acciones estratégicas.

Capítulo 4

Implementación

En este apartado se explicará brevemente la implementación del modelo expuesto en el capítulo anterior, centrándose en el diagrama de clases y las estructuras de datos que se han diseñado para dar soporte al modelo planteado.

4.1 Diagrama de clases

La clase básica que modela el comportamiento de las hormigas es la clase abstracta Ant:

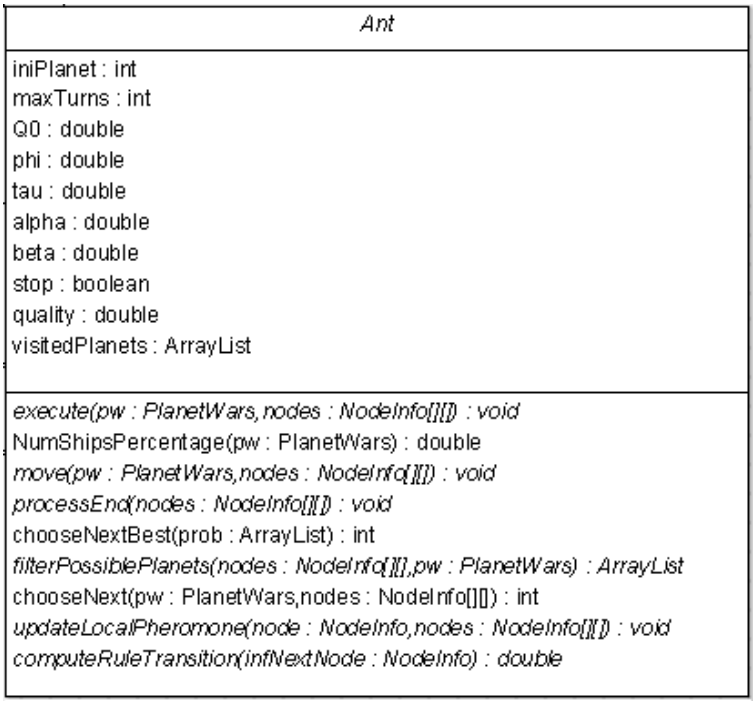


Figura 17: Clase Ant

Esta clase contiene la definición de las variables y métodos necesarios para implementar el funcionamiento de las hormigas, tal y como se definió en el modelo anterior.

Los atributos de la clase contienen información que la hormiga debe almacenar y/o actualizar a lo largo de su vida, y pueden estar relacionadas con el entorno que las rodea (el planeta de inicio, la lista de los planetas visitados, etc.) o con la parametrización de las reglas del propio sistema ACS:

- *iniPlanet*: el planeta de inicio del que parte la hormiga. Se considera automáticamente como un planeta ya visitado y la hormiga no puede volver a pasar por ahí ya que un planeta no puede enviar tropas a sí mismo, según las reglas del juego.
- *visitedPlanets*: la lista de planetas ya visitados. Así la hormiga puede recordar el camino realizado y evitar pasar dos veces por el mismo sitio o evaluar la solución final encontrada.
- *maxTurns*: la distancia máxima, en turnos, a la que puede viajar la hormiga. Este parámetro limita el número de nodos elegibles para ser visitados ya que sólo se consideran elegibles aquellos planetas que están a menos de maxTurns de distancia del planeta actual de la hormiga.
- *Q0, phi, tau, alfa* y *beta*: son las variables de parametrización del ACS y se utilizan para la inicialización de los valores de feromona en los arcos y para los cálculos de las reglas de transición y actualización de la feromona.
- *quality*: es el valor de la calidad de la solución encontrada por la hormiga al final de su movimiento.
- *Stop*: indica si se ha alcanzado la condición de parada y, por lo tanto, la hormiga ha finalizado su movimiento.

La clase Ant tiene además los siguientes métodos:

- *NumShipsPercentage*: Devuelve el ratio de tropas enemigas versus propias, para tener un indicador de la fuerza relativa de ambos jugadores.
- *chooseNext*: Elige el siguiente planeta (nodo) a visitar en función de la probabilidad de cada planeta elegible de ser escogido.
- *chooseNextBest*: Devuelve el planeta (nodo) con mayor probabilidad de ser escogido como el siguiente planeta a visitar.

El resto de los métodos son sobrescritos por las clases DefensorAnt y ConquerorAnt, que heredan de la clase Ant, y especifican el comportamiento propio de las hormigas de dos de los modelos descritos: defensivo y expansivo, respectivamente.

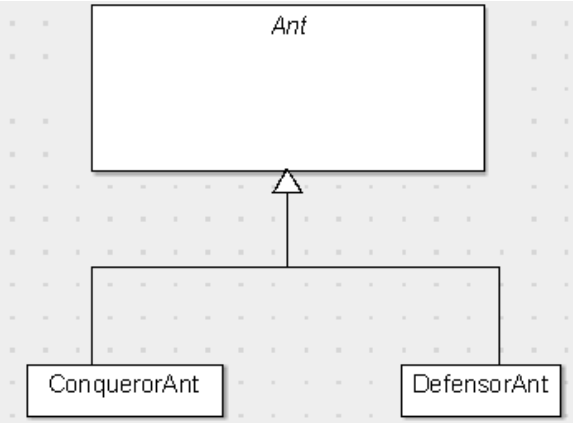


Figura 18: Clases Ant, DefensorAnt y ConquerorAnt.

A continuación se muestran las clases ConquerorAnt y DefensorAnt en detalle:

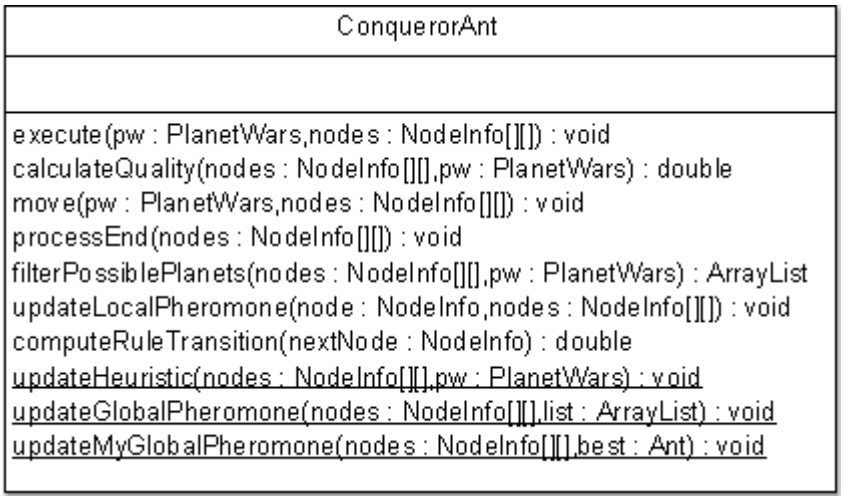


Figura 19: Clase ConquerorAnt

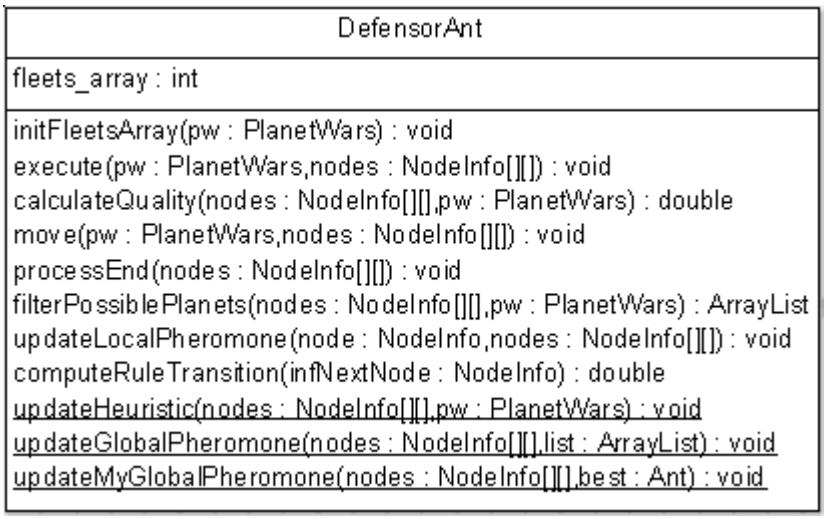


Figura 20: Clase DefensorAnt

Éstas son las clases que realmente implementan el algoritmo ACS aplicado al problema de Planet Wars, y aportan la inteligencia para que cada hormiga tome la decisión pertinente en cada salto en función de su objetivo. Los métodos que implementan son:

- *execute*: Ejecuta un movimiento de la hormiga, en un bucle, hasta que ésta alcance la condición de parada.
- *move*: Escoge el siguiente planeta al que va a ir la hormiga, de una lista de planetas elegibles y según las probabilidades de las reglas de transición, hasta que haya visitado todos los planetas posibles y tenga que parar. Una vez realizado el salto, se actualiza la feromona local, para favorecer la exploración del mapa.
- *filterPossiblePlanets*: Devuelve la lista de planetas elegibles. En el caso del modelo expansivo, se consideran elegibles los planetas enemigos o neutrales, que no hayan sido visitados aún (ver Fórmula 22). En el caso del modelo defensivo, se consideran elegibles los planetas propios, que no hayan sido visitados aun y que estén siendo atacados por el oponente (ver Fórmula 28).
- *computeRuleTransition*: este método implementa el cálculo de la regla de transición (ver Fórmula 23) usando las variables (feromona, α, β, etc.) de cada tipo de hormiga (expansiva o defensiva).
- *calculateQuality*: devuelve el valor de la calidad de la solución encontrada por la hormiga. Para ambos modelos, se mide el total de tropas aportadas en los próximos 100 turnos por todos los planetas visitados en la solución encontrada por la hormiga, teniendo en cuenta el Growth Rate de cada planeta, su coste inicial (sin tener en cuenta las acciones enemigas) y la distancia entre ellos (ver Fórmula 25). Como ya se ha mencionado anteriormente, la función de fitness se ha definido de la misma manera en la defensa que en la conquista ya que la defensa se plantea en ese caso como la conquista de unos planetas que de otro modo serían arrebatados por el contrincante.
- *updateLocalPheromone*: implementa la regla de evaporación de feromona para los modelos expansivo (ver Fórmula 24) y defensivo (ver Fórmula 31).
- *updateGlobalPheromone*: implementa la regla de actualización global de feromona para los modelos expansivo (ver Fórmula 26) y defensivo (ver Fórmula 32).

- *updateHeuristics*: una función demonio que actualiza a 0 la visibilidad de los planetas conquistados para las hormigas expansivas y de los planetas enemigos para las hormigas defensivas.

Para la mayoría de sus acciones las hormigas, modeladas por las clases ConquerorAnt y DefensorAnt, usan una estructura auxiliar que contiene información sobre el mapa del combate (grafo). Esta estructura auxiliar consiste en un Array de objetos NodeInfo, cada uno de los cuales representa un arco del grafo con la siguiente información (ver Fórmula 20):

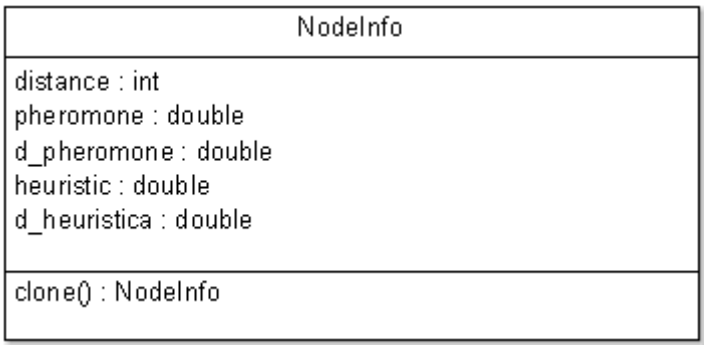


Figura 21: Clase NodeInfo

Los atributos de la clase NodeInfo son:

- *distance*: la longitud en turnos de este arco, es decir, la distancia entre los planetas unidos por el arco descrito por NodeInfo.
- *pheromone* y *d_pheromone*: el valor de feromona para ConquerorAnt y DefensorAnt, respectivamente, en el arco descrito por NodeInfo.
- *heuristic* y *d_heuristic*: el valor de la función heurística para ConquerorAnt y DefensorAnt, respectivamente, en el arco descrito por NodeInfo.

Además de las clases que modelan el comportamiento de la colonia de hormigas, se describen brevemente las clases que representan el resto del universo de Planet Wars.

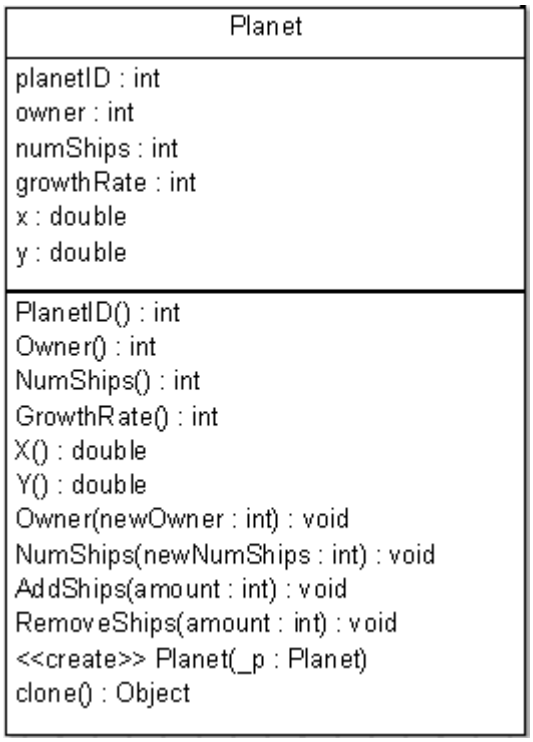


Figura 22: Clase Planet

La clase Planet representa un planeta dentro del dominio del problema. Sus principales atributos contienen la información que los jugadores necesitan saber sobre cada planeta:

- *planetID*: el identificador único del planeta.
- *owner*: indica si el planeta es propio, neutral o del enemigo.
- *numShips*: el número de tropas almacenadas en el planeta.
- *growthRate*: la tasa de crecimiento del planeta.
- *x* e *y*: las coordenadas del planeta en el mapa, que sirven para hallar la distancia entre dos planetas cualesquiera.

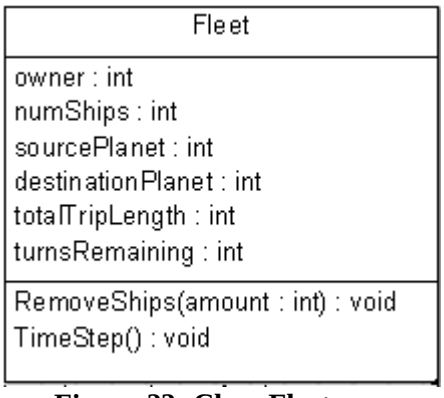


Figura 23: Clase Fleet

La clase Fleet modela una orden de envío de tropas en Planet Wars:

- *owner*: el jugador que envía la tropa.
- *numShips*: la cantidad de tropas que se manda en un único envío.
- *sourcePlanet*: el planeta origen desde donde se mandan las tropas.
- *destinationPlanet*: el planeta destino al que se mandan las tropas.
- *totaltripLength*: la distancia, en turnos, entre los planetas origen y destino.

- *turnsRemaining*: la distancia, en turnos, entre la posición actual de las tropas y su destino.

Los métodos principales de la clase Fleet son:

- *RemoveShips*: resta el número de tropas a enviar del planeta origen en el momento del envío.
- *TimeStep*: avanza las tropas la distancia correspondiente a cada turno, hasta llegar a su destino.

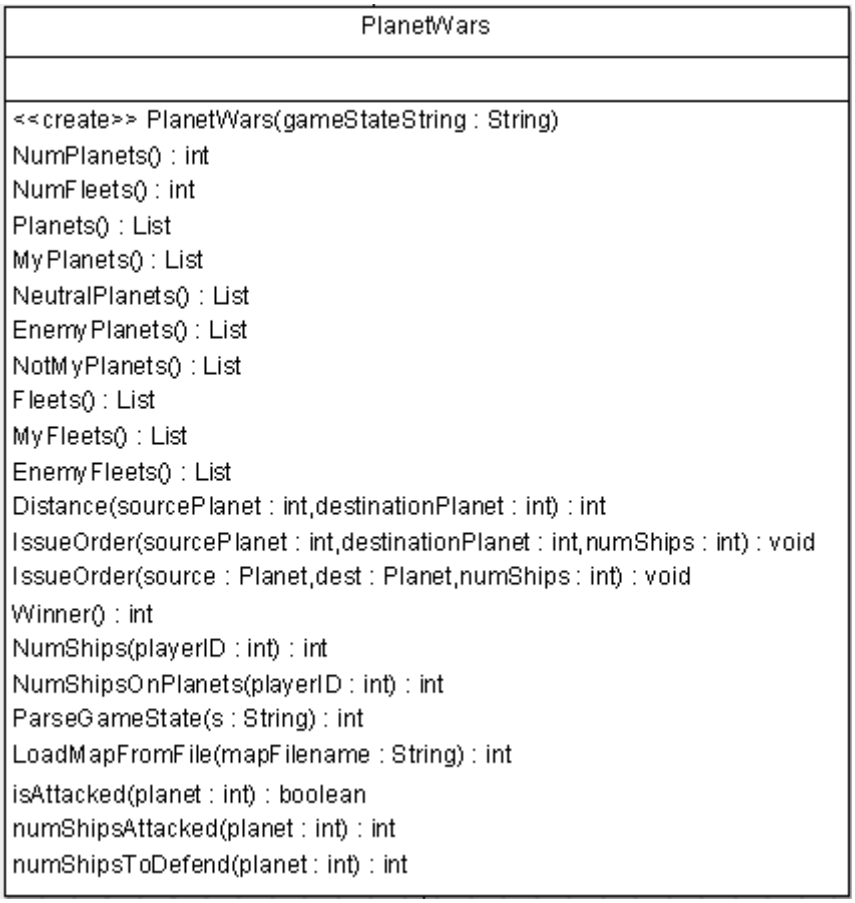


Figura 24: Clase PlanetWars

La clase PlanetWars contiene la información sobre el estado global de la partida:

- *NumPlanets*: indica el número total de planetas en el mapa de la partida actual.
- *NumFleets*: indica el número total de tropas en la partida actual.
- *Planets*, *MyPlanets*, *NeutralPlanets*, *EnemyPlanets*, *NotMyPlanets*: devuelven las listas de planetas total, de planetas propios, neutrales, del enemigo, o de todos los planetas que no sean propios (es decir, que son o bien neutrales o bien del enemigo), respectivamente.
- *Fleets*, *MyFleets*, *EnemyFleets*: devuelven las listas de tropas totales, propias o del enemigo, respectivamente.
- *Distance*: calcula la distancia entre dos planetas.
- *IssueOrder*: ejecuta la orden de enviar un número de tropas determinado de un planeta origen a un planeta destino.

- *Winner*: indica el jugador ganador de la partida, según las condiciones del juego de Planet Wars.
- *NumShips, NumShipsOnPlanet*: obtiene el número de tropas que posee un jugador en total, o en un planeta en particular.
- *isAttacked*, *numShipsAttacked*, *numShipsToDefend*: indican si un planeta está siendo atacado, con qué cantidad de tropas y cuántas tropas se necesitarían para defender el planeta en cuestión.
- *ParseGameState*, *LoadMapFromFile*: son métodos auxiliares que permiten escribir el estado de la partida en un fichero de logs o cargar el mapa correspondiente, para poder comenzar el combate.

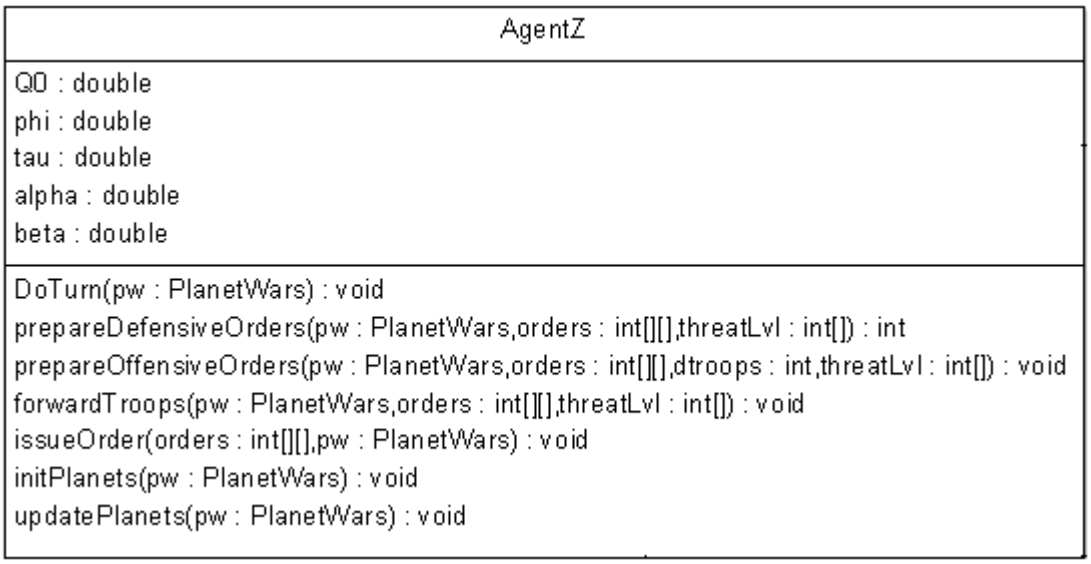


Figura 25: Clase AgentZ

La clase AgentZ modela al agente que va a jugar en la partida, incorporando todos los elementos descritos en el Capítulo 3. Sus atributos principales son las variables *Q0, phi, tau, alpha* y *beta*, que se pasan a la clase Ant en el momento de crear a las hormigas. Los métodos principales de la clase AgentZ son:

- *prepareDefensiveOrders*: este método crea un conjunto de hormigas defensivas, instancias de la clase DefensorAnt, que ejecutan sus acciones hasta que todas hayan alcanzado la condición de parada. A continuación, se crea un array de órdenes para enviar tropas a defender los planetas propios, según los resultados obtenidos por el ACS del modelo defensivo.
- *prepareOffensiveOrders*: este método es muy similar al anterior, con la diferencia de que las hormigas que se crean son hormigas ofensivas, instancias de la clase ConquerorAnt. Se ejecutan los pasos del algoritmo ACS hasta que todas las hormigas alcancen su condición de parada. A continuación, se crea un array de órdenes para enviar tropas a conquistar planetas neutrales o enemigos.
- *forwardTroops*: este método ejecuta las acciones del balanceo de tropas, una vez terminadas las acciones de conquista y defensa. Se asigna un índice de peligrosidad (IP) a cada planeta (ver Fórmula 33) y se crea un array de órdenes para enviar el sobrante de tropas desde planetas de menor IP hacia planetas de mayor IP.
- *issueOrders*: invoca el método issueOrders de la clase PlanetWars, para cada una de las órdenes creadas en los métodos anteriores.

- *initPlanets*, *updatePlanets*: son métodos auxiliares que inicializan y actualizan, respectivamente, la información sobre los planetas en juego, utilizando los métodos de la clase PlanetWars.
  DoTurn: es el método principal de la clase y ejecuta todas las acciones que debe realizar el agente cada turno.

```
If es el primer turno then initPlanets() else updatePlanets();
For all MyPlanets do
      PrepareDefensiveOrders();
      prepareOffensiveOrders();
end for
forwardTroops();
```

Figura 26: Pseudocódigo para el método DoTurn().

Por último, se incluye un diagrama de clases global, con el fin de mostrar las relaciones que se producen entre ellas:

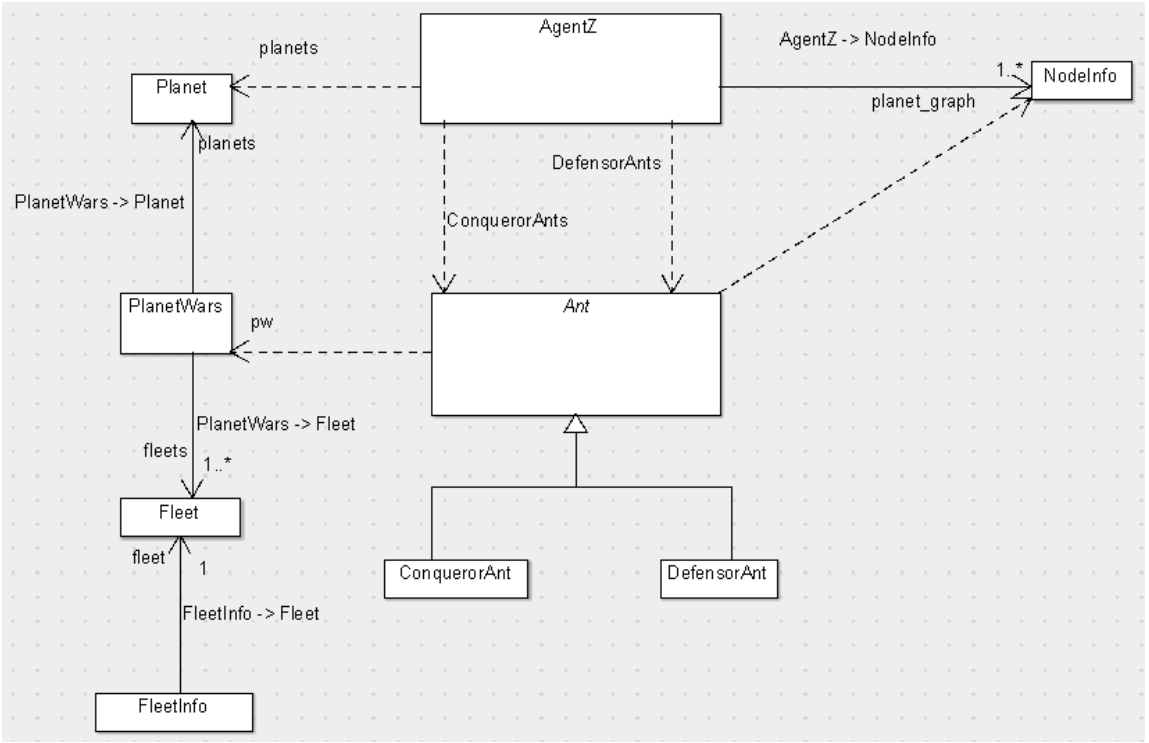


Figura 27: Diagrama de clases global.

Capítulo 5

Validación del Modelo: Experimentación

En este apartado se procederá a realizar diversas pruebas al agente implementado para validar el modelo planteado en el capítulo anterior.

La descripción de dichas pruebas se estructurará de manera similar a la presentación del modelo.

En primer lugar, se presentarán las pruebas para determinar la validez de cada una de las heurísticas planteadas a lo largo del desarrollo del problema, que se pueden ver en la Figura 13. A continuación, se describirán las pruebas específicas diseñadas para cada subproblema identificado en el capítulo anterior y se presentarán los resultados obtenidos mediante la evaluación de un muestreo de agentes. Por último, se incluirán pruebas adicionales, realizadas al bot completo, en distintos mapas o contra distintos contrincantes del concurso real, para ampliar lo máximo posible el estudio de su comportamiento y desempeño.

5.1 Evaluación de las heurísticas planteadas.

En la descripción del modelo se han planteado cuatro diferentes heurísticas, que ayudan a guiar la búsqueda de la solución por el agente en función de varios aspectos, como la distancia al planeta a conquistar, su coste en tropas o la tasa de crecimiento.

Para cada heurística, se han ejecutado 6 combates sobre 20 mapas, cada combate de un máximo 200 turnos. Para la evaluación de resultados, se ha recogido información sobre el número de victorias totales, la rapidez de la expansión del agente y el número de tropas que logra acumular.

5.1.1 Validación del modelo: Expansión.

5.1.1.1 Bot básico

En este primer conjunto de pruebas se han ejecutado combates del agente con un enemigo muy sencillo, que no hace ningún movimiento de conquista y se dedica a acumular tropas en su planeta. En estos combates, se evalúa sobre todo la capacidad de expansión del agente en condiciones ideales, es decir, sin contrincante. Debido a eso, no tiene sentido evaluar los resultados basándose en el número de victorias conseguidas, porque el agente siempre va a ganar ya que o bien conquista al contrincante en un movimiento de expansión que incluya su planeta como destino, o bien acumulará más tropas al final de la partida ya que tendrá más planetas en su poder.

Agentes a evaluar: 4
Enemigos: 1
Mapas: 20
Combates por mapa y agente: 6

Figura 28: Parámetros de pruebas de heurísticas planteadas (I).

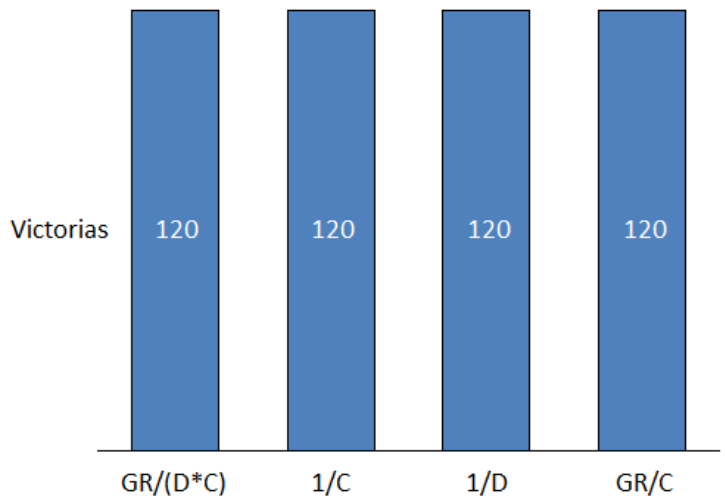


Figura 29: Victorias del modelo expansivo (I).

En esta gráfica se hace aparente que el número de victorias no es el indicador representativo para medir la capacidad del agente en esta fase del desarrollo. Para lograr obtener resultados significativos, se necesita tener en cuenta algún otro aspecto como la rapidez de la expansión (turnos) o el volumen del ejército acumulado (tropas).

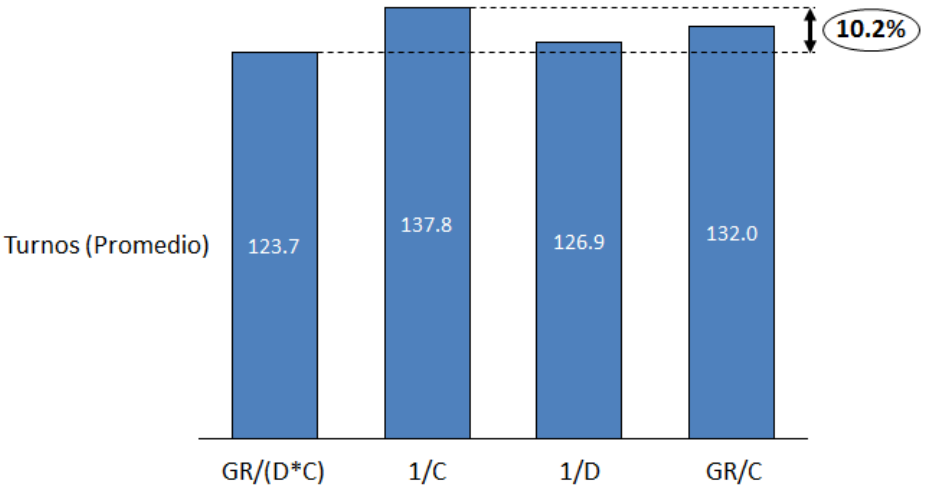


Figura 30: Turnos de combate del modelo expansivo (I).

Se aprecia que el agente con la primera de las heurísticas (GR/(D*C)) es el que menos turnos tarda en ganar, de media. Es decir, es la expansión más agresiva, hablando en términos de rapidez. En el lado contrario, se encuentra el bot que conquista a los planetas más baratos. Eso no siempre es rentable en tiempo, de hecho es el caso que más turnos tarda entre todos, ya que los planetas más baratos pueden encontrarse más lejos, por lo que las tropas enviadas tardarían muchos turnos en llegar. Esa diferencia se vuelve más radical cuando el enemigo tiene la capacidad de contraatacar. Los movimientos que implican la indisponibilidad de tropas durante largos períodos de tiempo son muy peligrosos y pueden significar la diferencia entre la victoria o la derrota.

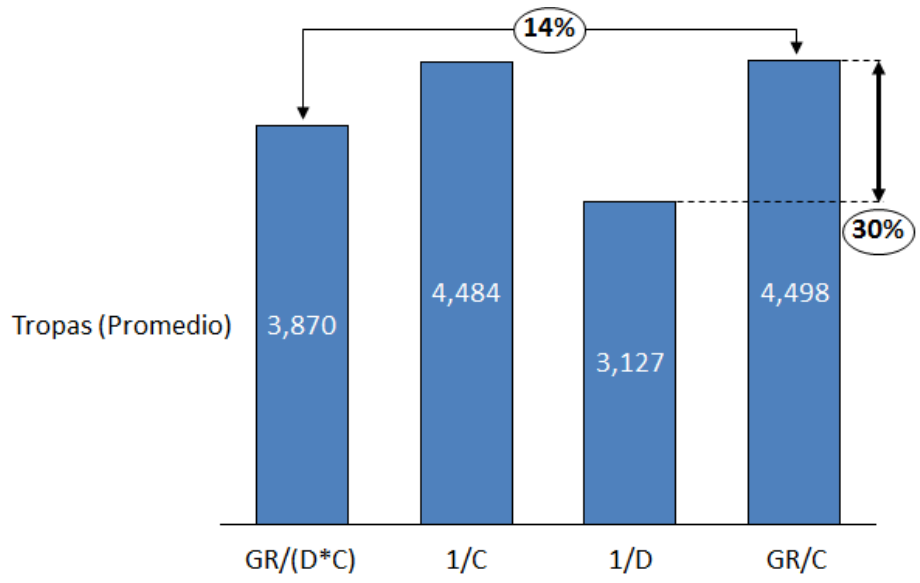


Figura 31: Tropas acumuladas del modelo expansivo (I).

La foto cambia evaluando las tropas conquistadas por cada agente. En este caso, el agente que conquista primero el planeta más barato, invierte menos tropas que las demás y, a la larga, dispone de un ejército mayor. El que menos tropas acumula, en este caso, es el bot que decide conquistar los planetas más cercanos. Puesto que el hecho de que un

planeta esté a menos distancia no significa que sea menos costoso, esta opción aparece como la menos rentable en términos de potencia de fuego acumulada.

En este punto, se puede ver que todas las heurísticas cumplen con el objetivo principal del juego, que es ganar al oponente. Es cierto que el enemigo elegido representa un reto más bien sencillo, por lo es pronto para sacar conclusiones definitivas. Sin embargo, sí se puede observar diferencias en los resultados obtenidos una vez acabado el combate. Las heurísticas más rápidas en el tiempo son, por lo general, las que menos tropas obtienen. Por el contrario, aquellas que se centran en acumular tropas terminan siendo ineficientes en el tiempo. La heurística que evalúa todos esos factores (GR/(D*C)) es la que ha obtenido los mejores resultados en cuanto a la duración de la partida y, a su vez, no ha quedado demasiado desposicionada en cuanto al tamaño su ejército.

5.1.1.2 Bots del starter Package

En esta fase de pruebas se ejecutarán combates del agente con el modelo expansivo completamente implementado, contra enemigos más complejos que en el caso anterior, que venían propuestos como bots de entrenamiento en el pack inicial del concurso Google AI Challenge. Dichos enemigos participan activamente en la conquista y defensa de sus planetas, aunque sus estrategias presentan alguna limitación:

- **DualBot**: mantiene un número constante de tropas en el vuelo. Cuando este número disminuye, porque las tropas hayan llegado a su destino, el bot lanza una orden para mandar más tropas en el próximo turno. Las tropas siempre salen del planeta propio más fuerte hacia el planeta enemigo más débil.
- **RageBot**: ataca únicamente los planetas del oponente y nunca los planetas neutrales. Primero selecciona todos los planetas que tengan un mínimo de tropas y a continuación manda todas sus tropas desde cada uno de esos planetas al planeta enemigo más cercano.
- **ProspectorBot**: tiene el mismo comportamiento que DualBot, salvo que sólo puede haber como máximo una tropa en vuelo. El bot no emite la orden de mandar la siguiente tropa hasta que la anterior haya llegado a su destino.
- **BullyBot**: ataca el planeta enemigo más fuerte. Primero elige el planeta propio con más número de tropas, para luego atacar el planeta enemigo con la mitad de sus tropas.

Igualmente, las pruebas consisten en ejecutar 6 combates con cada enemigo y heurística, en cada uno de los 20 mapas. Se recogerán los resultados del número de victorias y derrotas, la duración de la partida en turnos y las tropas acumuladas al final de la misma.

Agentes a evaluar: 4
Enemigos: 4
Mapas: 20
Combates por mapa y agente: 6

Figura 32: Parámetros de pruebas de heurísticas planteadas (II).

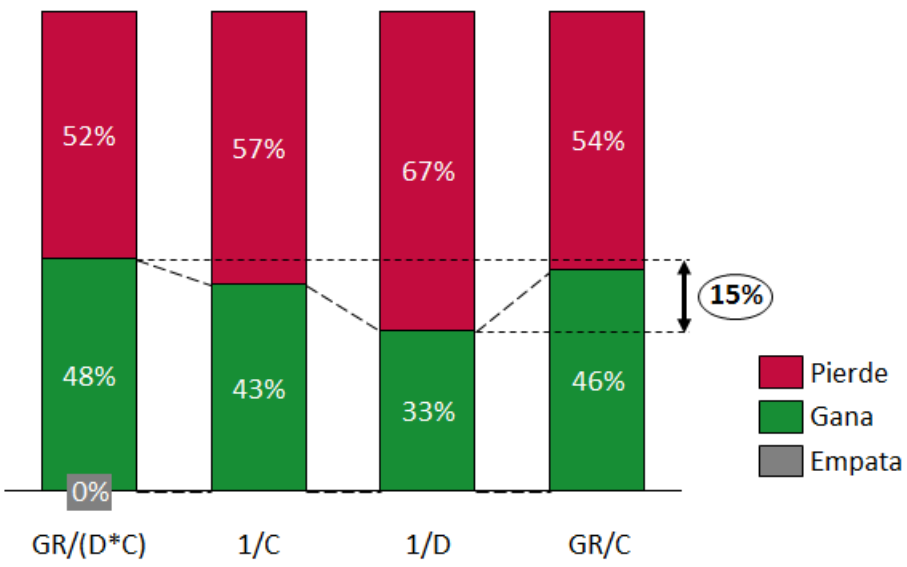


Figura 33: Victorias modelo expansivo (II)

Se aprecia cómo, ante bots más complejos, el modelo expansivo ya no es capaz de ganar todos los combates. El agente resulta incompleto, no teniendo capacidad de defenderse o acumular tropas en sitios estratégicos. Aun así, sólo centrándose en la conquista de los planetas es capaz de ganar entre una tercera parte y la mitad de los combates, en el peor y mejor caso, respectivamente. Mirando más detalladamente el desempeño de cada heurística, se observa que las más efectivas han sido las que tienen en cuenta la tasa de producción del planeta. Al tomar en cuenta más factores que las dos restantes, son capaces de tomar una decisión más informada y, por ende, más acertada, lo cual se traduce directamente en la competitividad del agente desarrollado. Al igual que en el apartado anterior, se hará una comparativa más exhaustiva, tomando en cuenta otros factores que intervienen en la batalla.

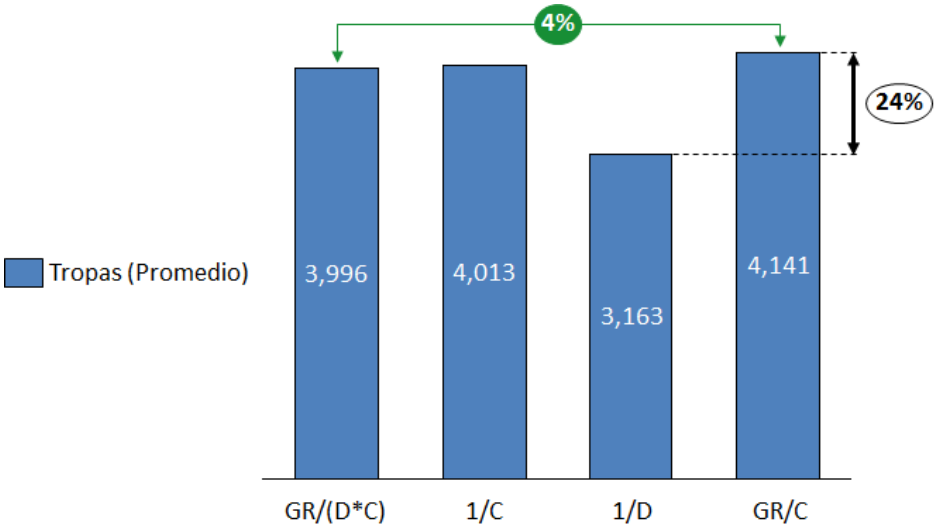


Figura 34: Tropas acumuladas del modelo expansivo (II)

Si se compara este resultado con el obtenido en las mismas pruebas del apartado anterior (Fig. 31), se puede ver que el desempeño de tres de las heurísticas se iguala bastante, reduciéndose la diferencia entre la primera y la cuarta del 14% a 4%. Esto se debe a que, peleando contra un enemigo más activo, el rendimiento de la cuarta heurística, que en el otro caso se mostraba como la mejor, baja susceptiblemente, mientras que el de la primera aumenta ligeramente. La heurística que obtiene peores resultados, en este caso, sigue siendo la que elige el planeta más cercano (1/D).

Realizando la comparativa por contrincante, salta a la vista que los bots que más dificultades plantean en el combate son el DualBot y ProspectorBot. Son los dos bots que tienen en cuenta tanto la fuerza de los planetas propios como enemigos, por lo que el agente necesitará mejorar su estrategia para ganar más partidas. De hecho, los resultados presentan una diferencia de un 97% en cuanto al número de victorias entre el bot más sencillo (RageBot) y el más complejo:

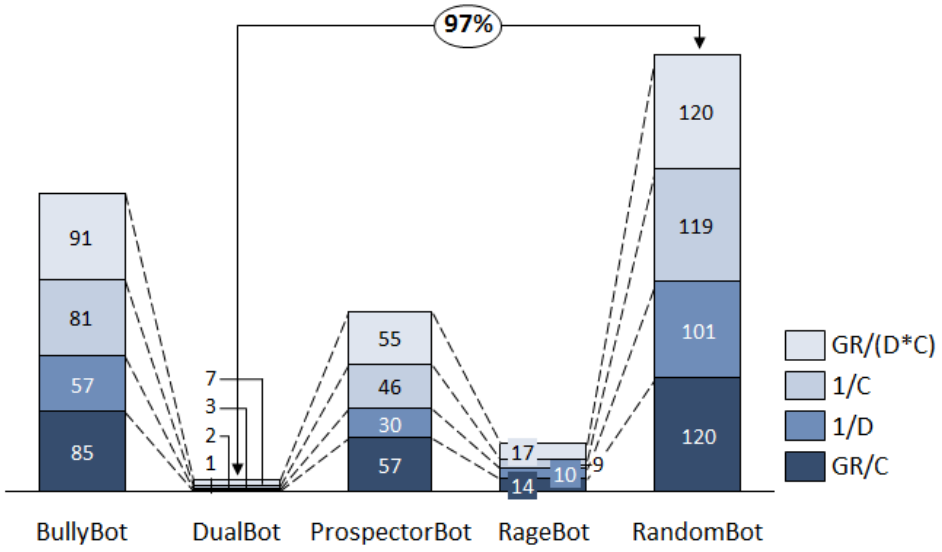


Figura 35: Comparativa por Bot del modelo expansivo.

Analizando la misma foto, pero cambiando los ejes de la gráfica anterior, se aprecia que siguen teniendo mejores resultados, en combates con todos los enemigos, aquellas heurísticas que tienen en cuenta el coste y el beneficio de un planeta, en cuanto a tropas se refiere:

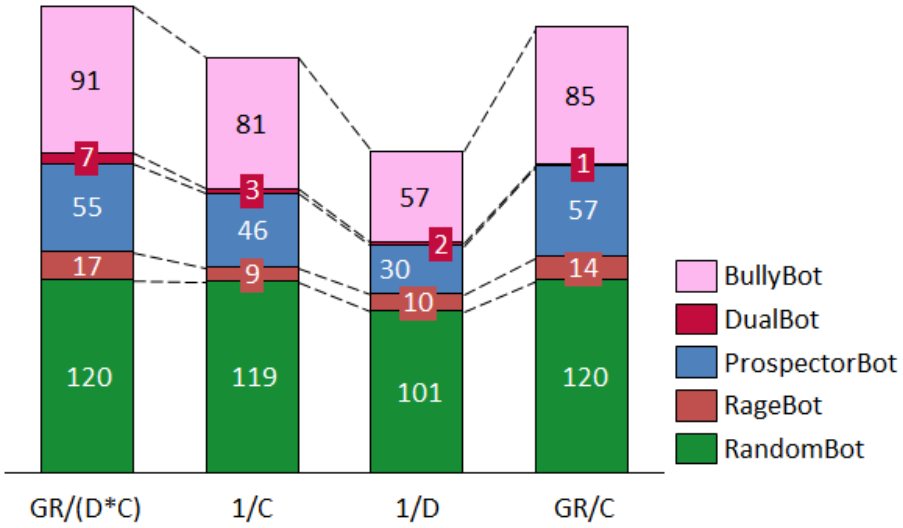


Figura 36: Comparativa por Heurística del modelo expansivo.

Además del tipo de enemigo, hay otro factor que puede influir en el desempeño del agente, y es la distribución de los planetas en el mapa. Viendo los resultados de enfrentamientos en distintos mapas, se observa la siguiente situación:

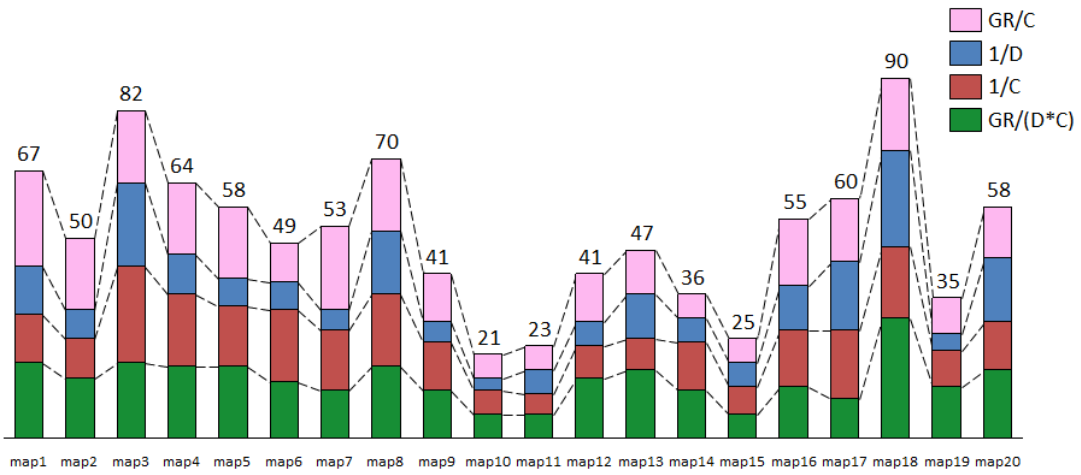


Figura 37: Comparativa por mapa del modelo expansivo.

Los mapas en el que el agente obtiene mejores resultados en general son el 18, 3 y 8. ¿Qué características comparten esos mapas para facilitar que el agente se desenvuelva con éxito?

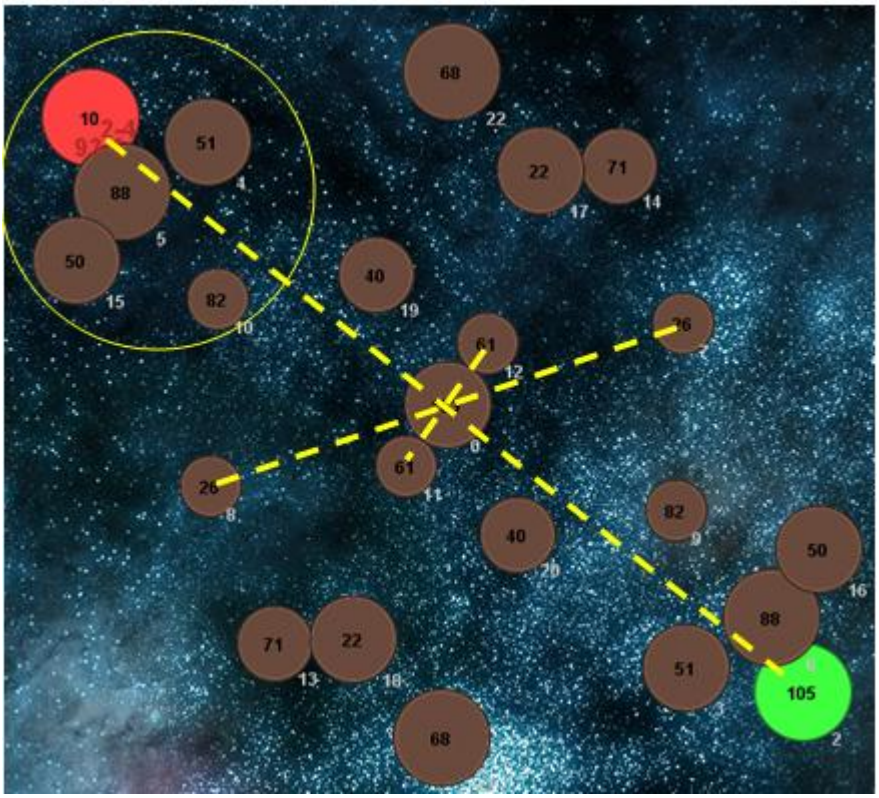


Figura 38: Mapa 18

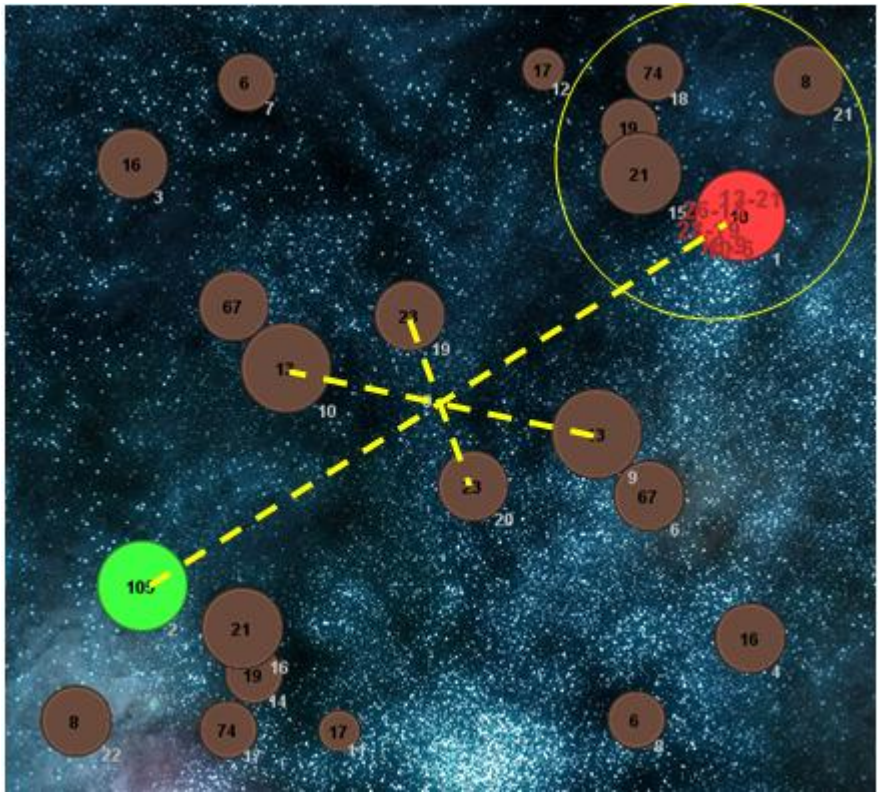


Figura 39: Mapa 3

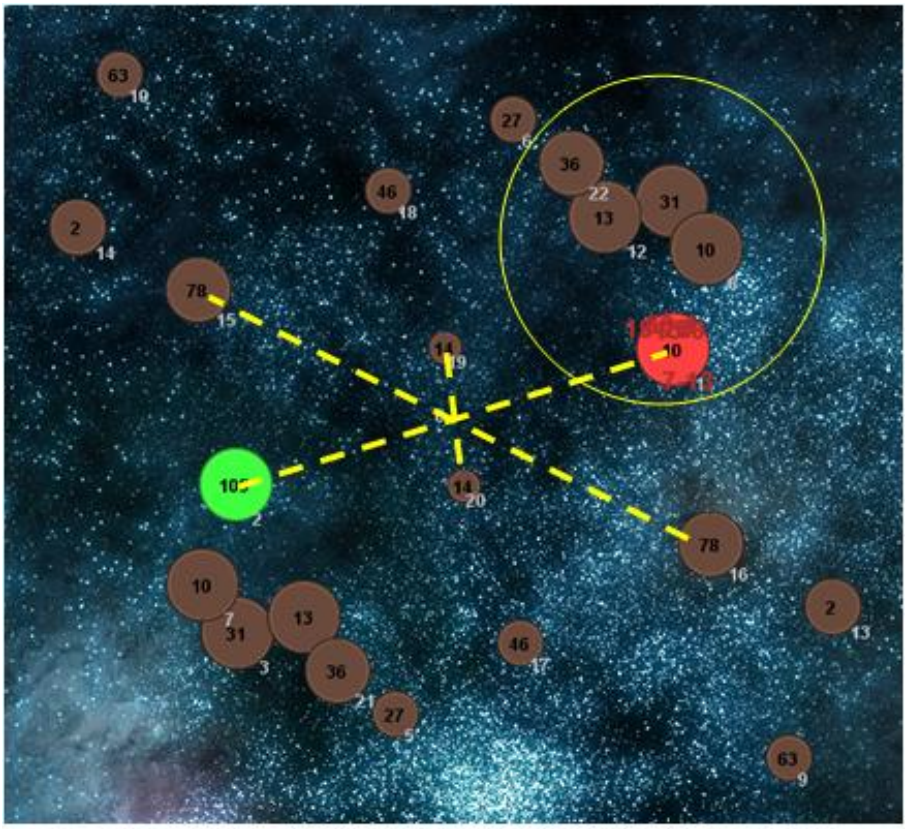


Figura 40: Mapa 8

Se observa que todos estos mapas tienes simetría radial, con respecto al punto central del mapa, en el que puede haber o no un planeta. El centro de la simetría está indicado por el cruce de los ejes que unen varios planetas entre sí, señalados con líneas amarillas punteadas.

En segundo lugar, salta a la vista que en todos los casos, los planetas origen están rodeados de planetas con alta tasa de crecimiento, que se encuentran a una distancia muy cercana. Esta situación de inicio es beneficiosa para el comienzo del agente, ya que permite que éste conquiste planetas rápidamente, aumentando la producción de tropas casi desde el primer momento. Además, también es ventajoso, en cierto sentido, que el enemigo haga lo propio, ya que eso "entretiene" al contrincante y retrasa el momento del conflicto entre ambos.

Por último, analizando la posición de ambos planetas origen entre sí, se aprecia que los jugadores comienzan a una distancia relativamente grande entre sí: bien planetas ubicados directamente en los extremos, como en el caso del mapa 18, o bien rodeados de un gran número de planetas neutrales más cercanos que el enemigo, como en el caso de los mapas 3 y 8. El efecto de esta distribución geográfica es el mismo que el mencionado en el párrafo anterior: los contrincantes se alejan entre sí, concentrando sus esfuerzos en planetas más apetecibles y cercanos, retrasando el conflicto.

Por el contrario, los mapas que peores resultados han obtenido son el 10, 11 y 15.

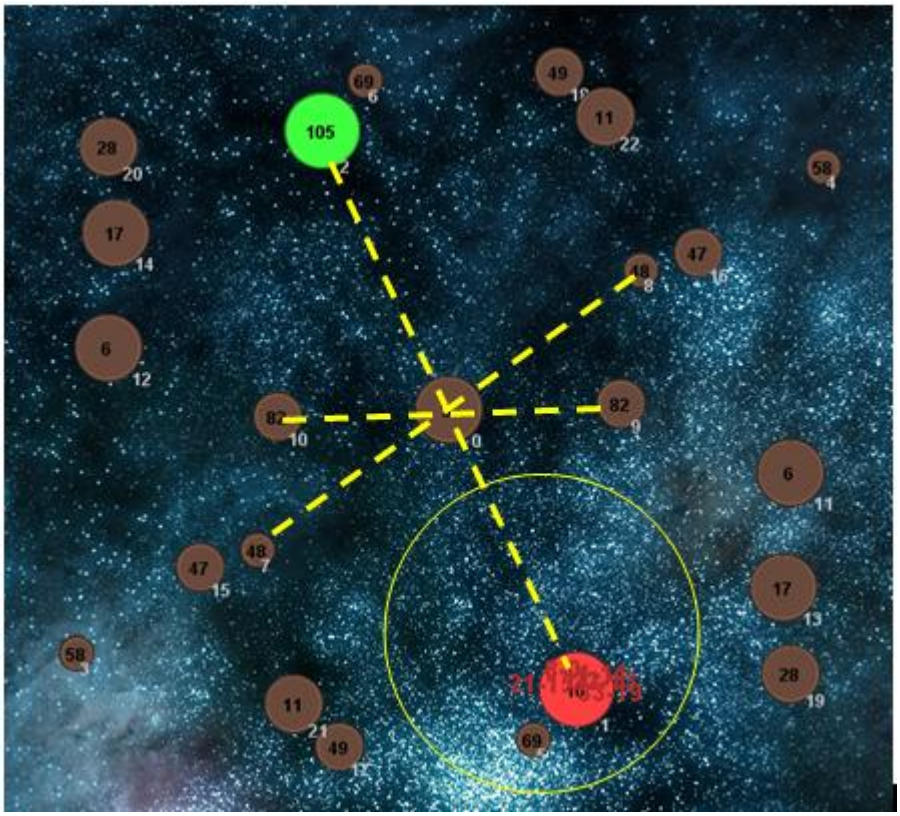


Figura 41: Mapa 10

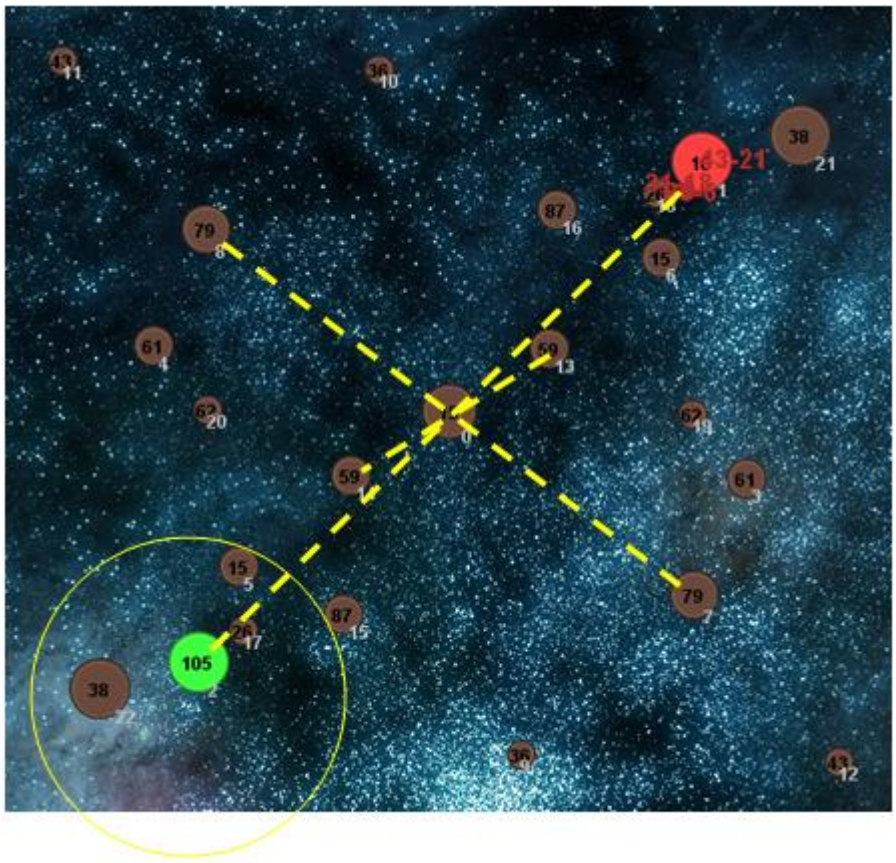


Figura 42: Mapa 11

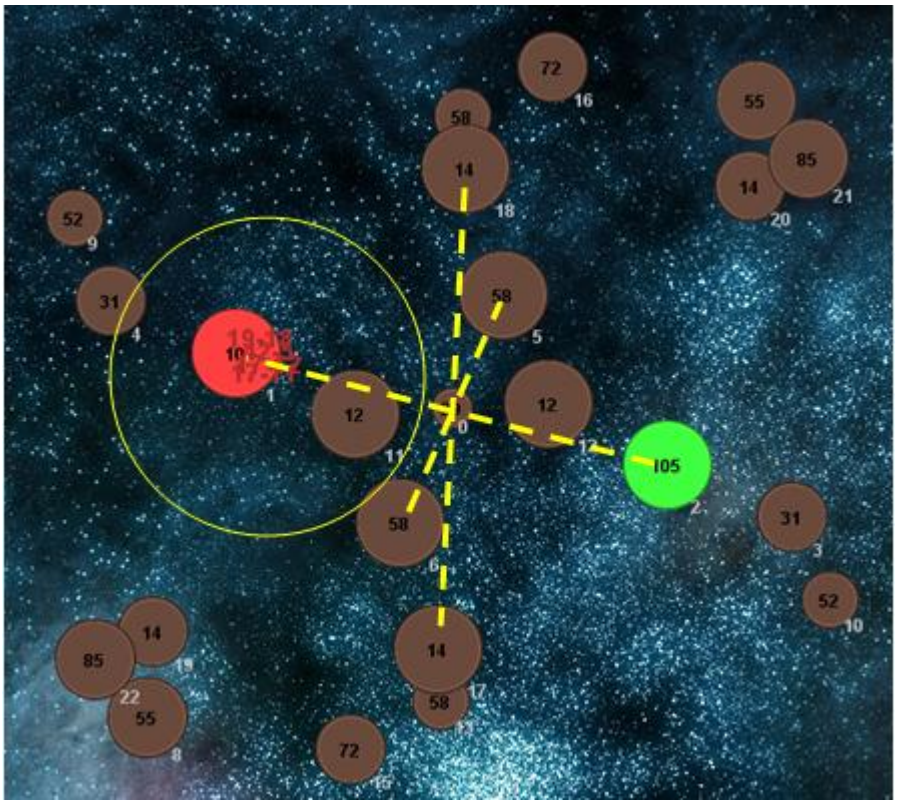


Figura 43: Mapa 15

Los mapas 10, 11 y 15, a su vez, también presentan una simetría central, por lo que se deduce que éste no es un factor determinante del comportamiento del agente. No obstante, ofrecen una distribución totalmente contraria a los mapas 18, 3 y 8 en cuanto a los otros dos factores se refiere. En cada mapa se ha trazado un círculo alrededor del planeta origen del agente, del mismo tamaño en todos ellos. Se puede observar que la cantidad de planetas neutrales cercanos al inicial, es decir, de planetas que entran dentro de ese círculo señalado, es mucho menor en los mapas 10, 11 y 15. Además, en los mapas 10 y 11 son planetas costosos de conseguir y con poca tasa de crecimiento, lo que no les hace en absoluto candidatos "apetecibles", al contrario de lo que pasaba con otros mapas.

En segundo lugar, la distancia entre los planetas origen de los enemigos al comienzo de la partida se reduce, sobre todo si se examina el mapa 15. En dicho mapa no sólo ambos planetas están cerca, sino que los planetas más favorables están justo entre ellos, a la misma distancia entre los dos.

Esta distribución geográfica tiene el efecto contrario al de los mapas 18, 3 y 8. En otras palabras, acelera el conflicto entre ambos bandos, obligándolos a luchar por los recursos más apetecibles desde el comienzo de la partida.

En conclusión, se puede afirmar que el desempeño del agente es mejor en aquellas situaciones que favorecen la expansión y acumulación de tropas, sin entrar en el combate propiamente dicho. Por el contrario, aquellas circunstancias que fuercen el choque con el enemigo, son las menos favorables para el agente. Esto es totalmente lógico, ya que en esta fase el agente tiene únicamente implementado el modelo de expansión y no está preparado para la complejidad que un enfrentamiento supone, lo cual vuelve a acentuar, una vez más, la importancia del siguiente módulo: la Defensa.

5.1.2 Validación del modelo: Defensa.

5.1.2.1 Bots del starter Package

Es el segundo módulo planteado para el agente, que complementa al primero, sobre todo en situación de combate con el enemigo. Para validar ese modelo, se ejecutarán el mismo conjunto de pruebas que con el modelo expansivo: 6 combates con cada enemigo y heurística, en cada uno de los 20 mapas. Se recogerán los resultados del número de victorias, derrotas y las tropas acumuladas al final de la misma. Se espera ver un incremento de victorias con respecto a la versión anterior del agente, que se centraba meramente en la conquista.

Agentes a evaluar: 4
Enemigos: 4
Mapas: 20
Combates por mapa y agente: 6

Figura 44: Parámetros de pruebas de heurísticas planteadas (III).

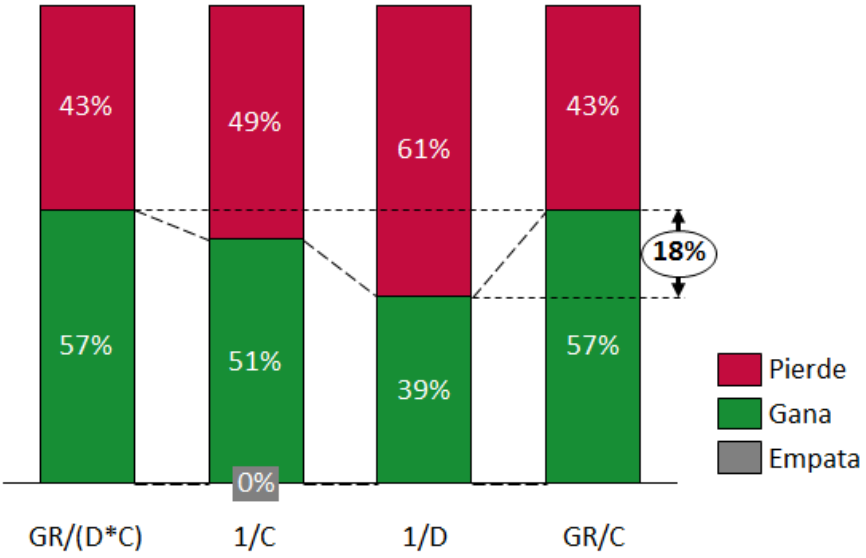


Figura 45: Victorias del modelo defensivo.

Se aprecia un incremento de victorias para todas las heurísticas con respecto a los resultados anteriores (ver Fig. 33). Par más detalle, se muestra a continuación el detalle del incremento que ha supuesto la inclusión de la estrategia defensiva:

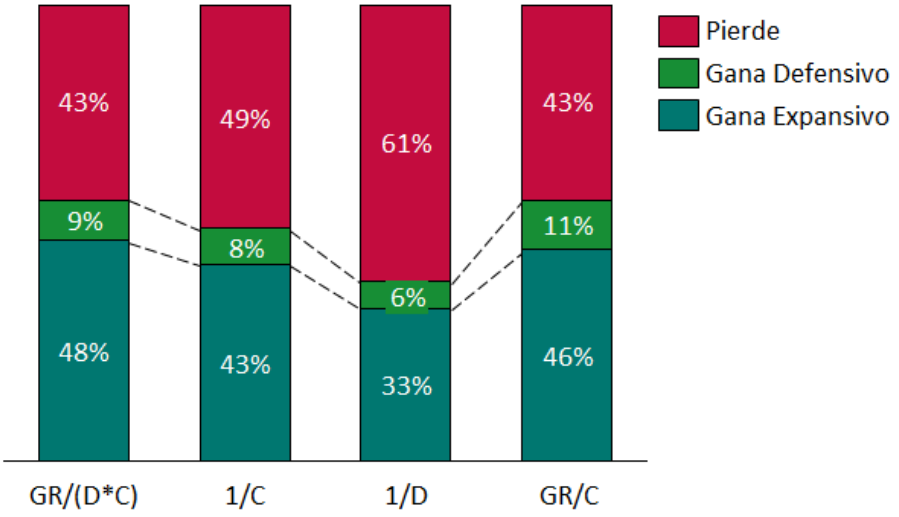


Figura 46: Incremento victorias modelo defensivo.

El Mayor impacto se aprecia en la heurística de relación GR/C, que ahora equipara sus resultados a la GR/(D*C). El agente que escoge los planetas más cercanos es el que menos ha mejorado su número de victorias, aumentando la distancia con los mejores agentes de un 15% a un 18%. Sin embargo, la distribución relativa de los resultados obtenidos por las distintas versiones del agente no cambia con respecto a los resultados del modelo expansivo puro.

En cuanto a la distribución por tipo de contrincante, se observa un aumento de victorias con todos los enemigos, incluidos el DualBot y el RageBot. En enfrentamientos con estos últimos, la diferencia con el mejor caso posible se reduce en 14p.p., de un 97% a un 83%. Destacan unos resultados casi perfectos contra el Random Bot, donde el agente en casi todos los casos termina con 120 combates ganados de 120 jugados.

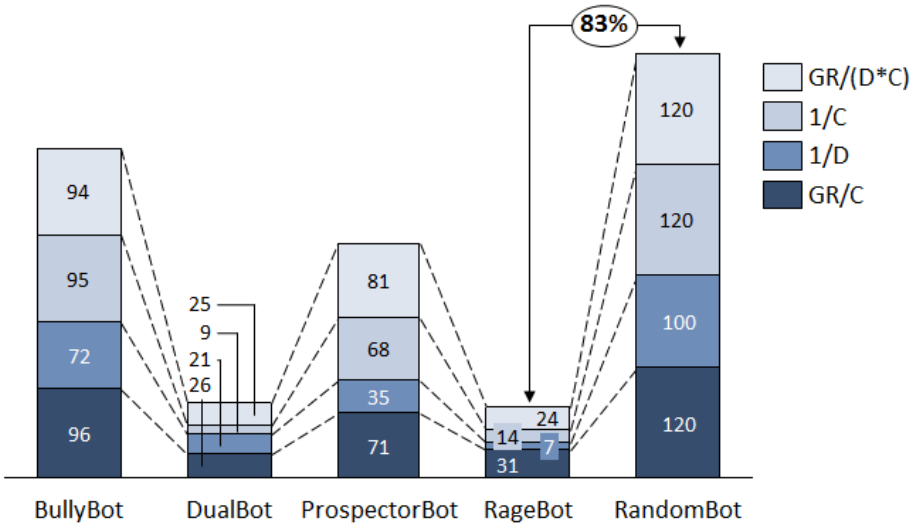


Figura 47: Comparativa por Bot del modelo defensivo.

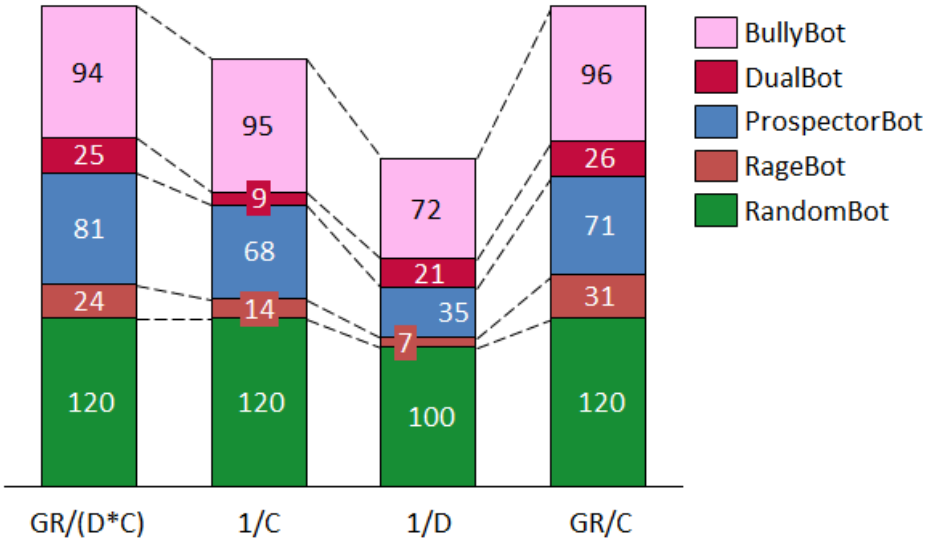


Figura 48: Comparativa por Heurística del modelo defensivo.

Por último, se realiza la comparativa por mapa, para analizar el impacto que puede tener la distribución geográfica de los planetas y tropas de ambos bandos en los resultados del modelo defensivo. Al igual que en el caso de resultados por tipo de enemigo, la distribución relativa de victorias en función del mapa no ha cambiado. No obstante, sí se aprecia un aumento de combates exitosos en diversos mapas.

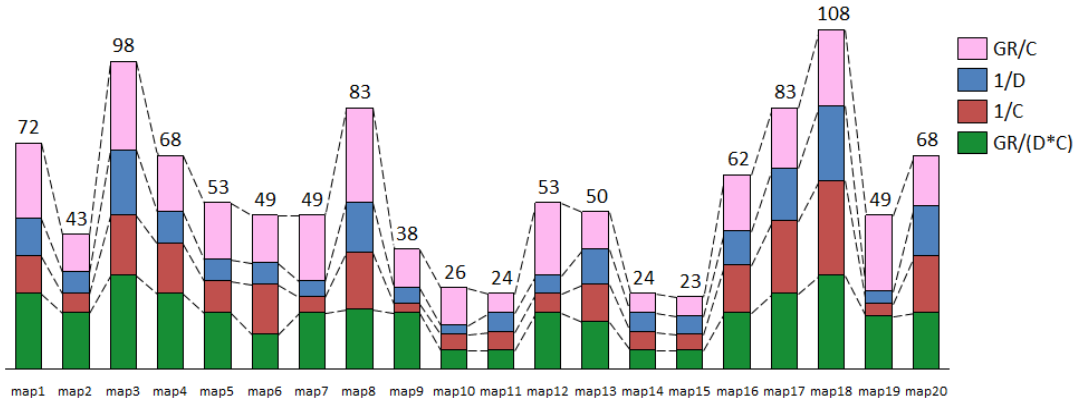


Figura 49: Comparativa por mapa del modelo defensivo.

Los mapas 18,3 y 8, que ya de por sí eran favorables en la fase de la conquista, han aumentado sus victorias en un 15% de media.

Los mapas 10,11 y 15 también han experimentado una mejora notable, aunque siguen siendo los más difíciles de resolver para el agente.

Además de los mapas ya mencionados, es interesante señalar un aumento de victorias en algunos casos particulares, que no se habían identificado anteriormente.

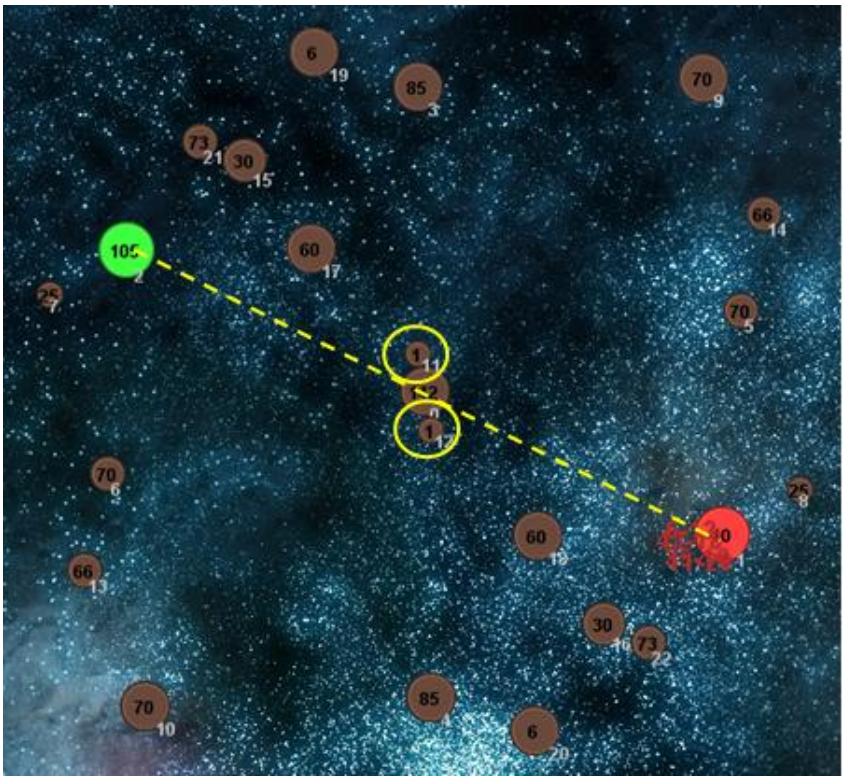


Figura 50: Mapa 13

Se aprecia una evolución positiva en los resultados de la heurística 1/C, que escoge los planetas menos costosos para conquistar primero, en el mapa 13. Este mapa presenta un claro punto de conflicto, que son los planetas 11 y 12 que claramente son los más baratos del mapa. Al no tener en cuenta ni la distancia ni la tasa de crecimiento, son los primeros planetas que intenta conquistar. Sin embargo, estos planetas están muy próximos al centro del mapa, convirtiéndose rápidamente en puntos "calientes" del combate. Por lo tanto, toda o mucha acción se centra en estos planetas y es muy importante defenderlos para no terminar perdiéndolos. Por ello, al implementar la estrategia defensiva, aumentan las victorias del agente con este mapa y esta heurística.

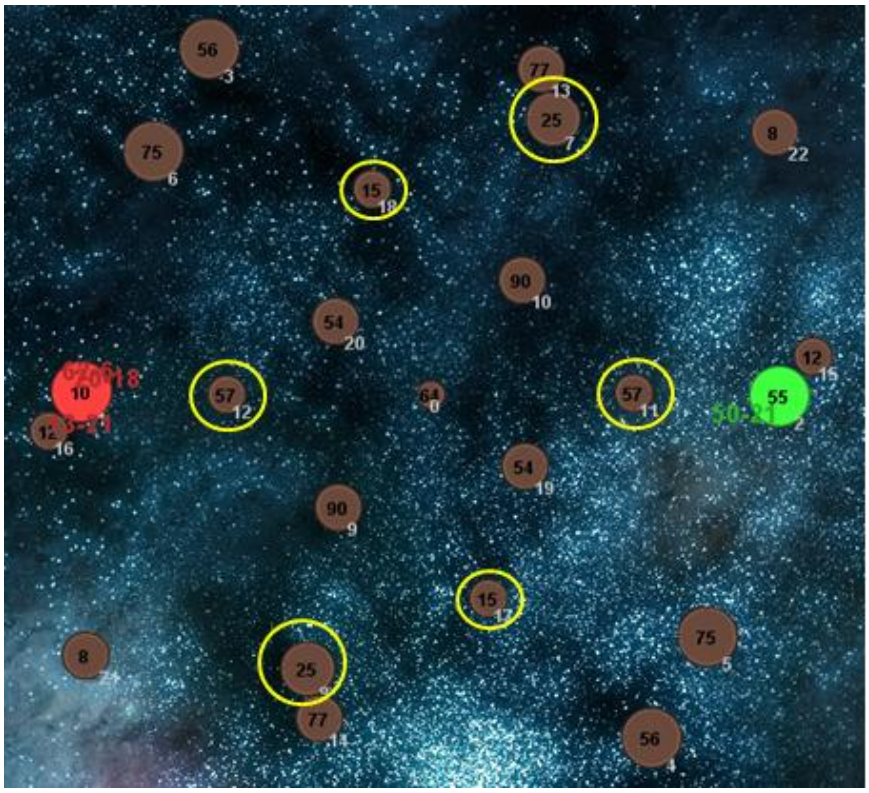


Figura 51: Mapa 12

Este caso es similar al anterior. El mapa es complicado, ya que la mayoría de los planetas son caros y generan pocas tropas (su tamaño en el mapa es pequeño). Además, están muy distanciados entre sí, lo cual ralentiza el proceso de conquista. Todo ello hace que una mínima diferencia en la rentabilidad (relación crecimiento/precio) aporte una ventaja competitiva determinante. Por ello, los planetas que sean incluso una ínfima fracción más apetecibles presentan un foco de batalla para ambos jugadores, por lo que una buena defensa cobra mayor relevancia y aporta mayor número de victorias.

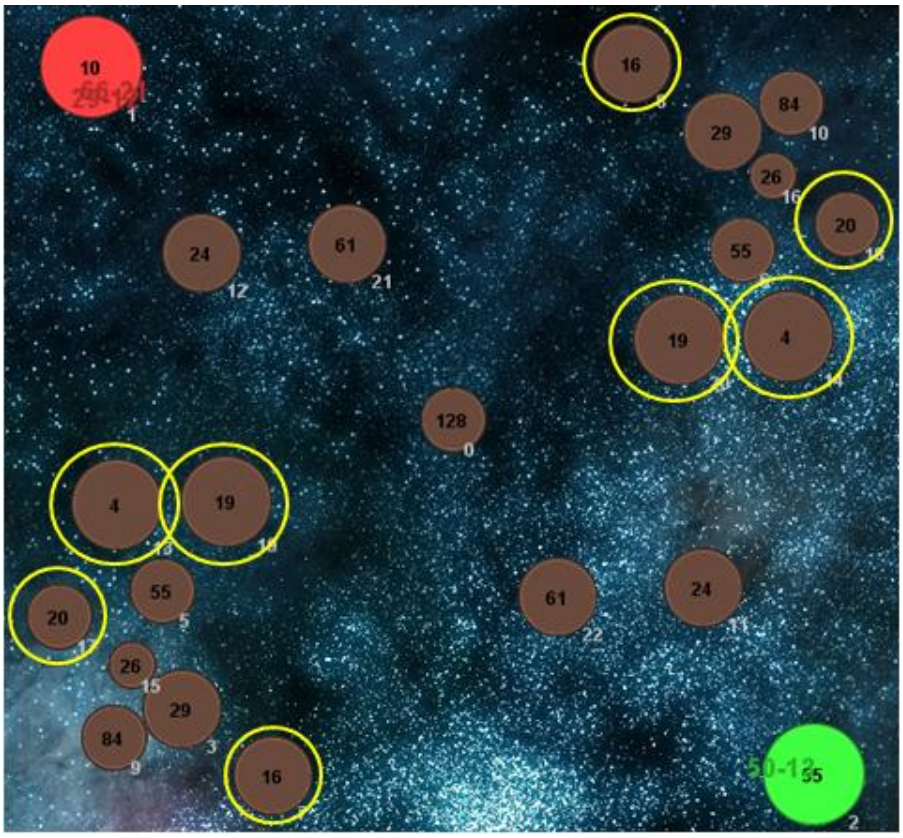


Figura 52: Mapa 19

Un caso parecido se produce en el mapa 19, aunque ahora en lugar de estar rodeados de planetas caros y pequeños, los jugadores están en medio de numerosos planetas grandes y baratos. Al tener ambos jugadores acceso a recursos abundantes, incluso una mínima diferencia en la rentabilidad aporta la ventaja competitiva que puede inclinar la balanza hacia un bando u otro, desembocando así en la misma situación que en el mapa anterior.

5.1.3 Validación del modelo: Balanceo de tropas.

El último módulo desarrollado es el de balanceo de tropas, que completa el conjunto de estrategias de batalla del agente propuesto. Tal y como se explica en el apartado 3.2.3 Balanceo de Tropas, el objetivo de este modelo es “pasar” las tropas disponibles en planetas más protegidos hacia planetas en la línea del frente con el fin de mejorar su disponibilidad para acciones de ataque o defensa.

5.1.3.1 Bots del starter Package

Para validar ese modelo, se ejecutará el mismo conjunto de pruebas que con los anteriores: 6 combates con cada enemigo y heurística, en cada uno de los 20 mapas. Se recogerán los resultados del número de victorias, derrotas y las tropas acumuladas al final de la misma. Se espera ver un incremento de victorias con respecto a la versión anterior del agente, que se centraba en la conquista y la defensa.

Agentes a evaluar: 4
Enemigos: 4
Mapas: 20
Combates por mapa y agente: 6

Figura 53: Parámetros de pruebas de heurísticas (IV).

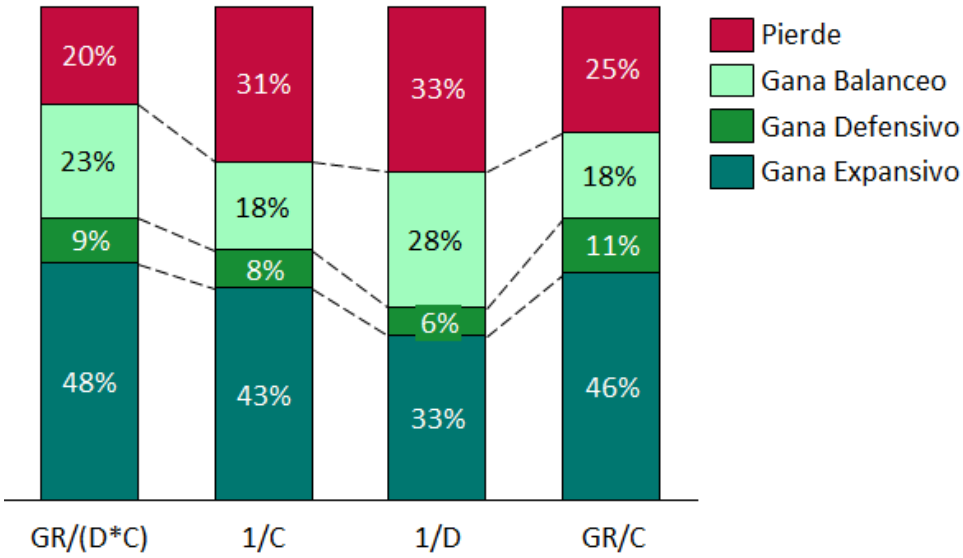


Figura 54: Incremento victorias balanceo de tropas

Como se puede observar en la figura anterior, los resultados obtenidos han sido muy positivos, respaldando la introducción del balanceo de tropas en la estrategia del combate. El porcentaje de victorias aumenta entre un 18% y un 28%, dependiendo de la heurística. La que obtiene los mejores resultados es GR/(D*C) y acumula en total un 80% de victorias, casi el doble del valor inicial obtenido.

Si se analizan los resultados por tipo de enemigo, las mejoras más notables se han experimentado contra los dos bots que más complicados han sido de vencer hasta ahora: DualBot y RageBot. Aunque su posición relativa en cuanto a número de derrotas con respecto a otros enemigos no cambia, sí se reduce la diferencia respecto de RandomBot de un 83% a un 65%.

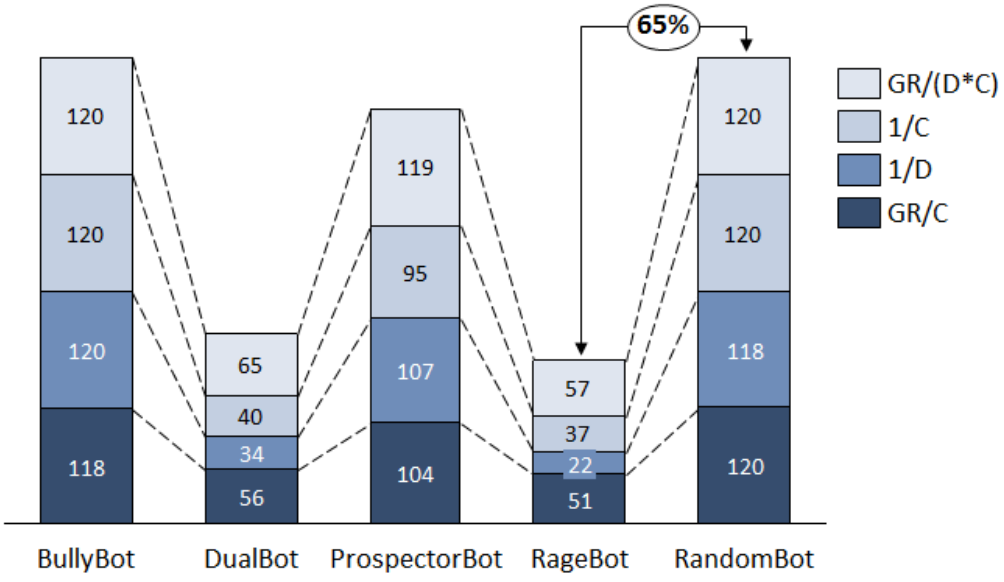


Figura 55: Comparativa por bot balanceo de tropas.

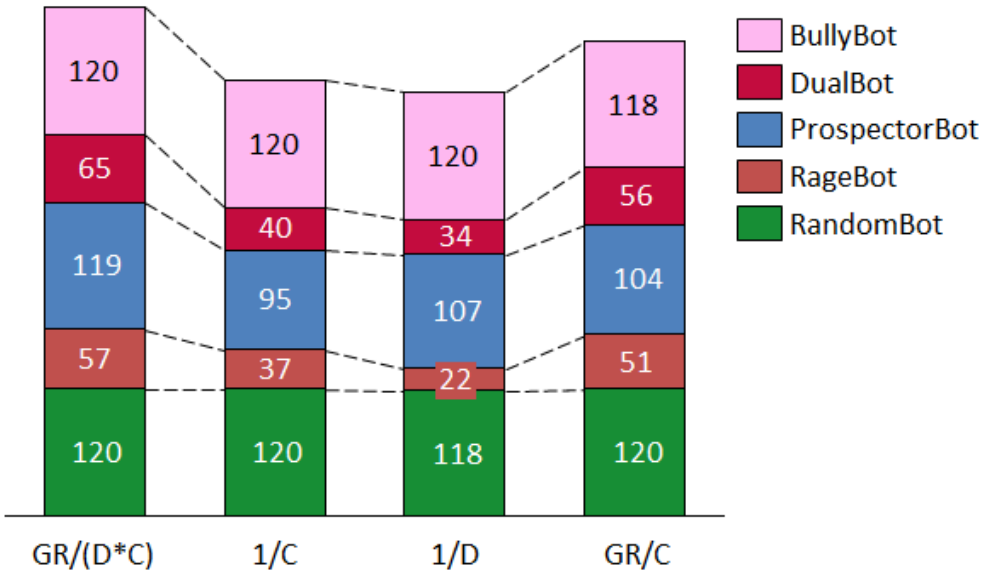


Figura 56: Comparativa por Heurística balanceo de tropas.

La comparativa por mapa es la siguiente:

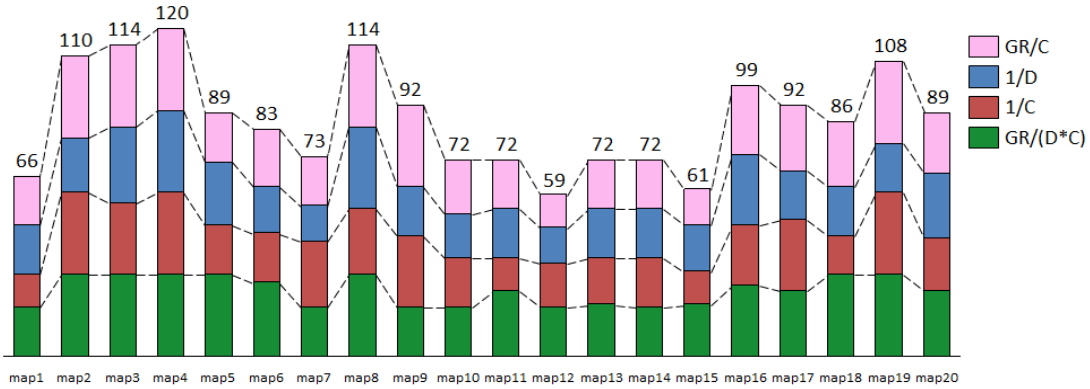


Figura 57: Comparativa por mapa balanceo de tropas.

Los que más han subido son los mapas 2 (250% aprox.), 10, 11, 14 y 15 (un 300% aprox.). Sin embargo, el mapa 19 ha bajado de 108 a 49 victorias, afectando principalmente a las heurísticas 1/D y 1/C.

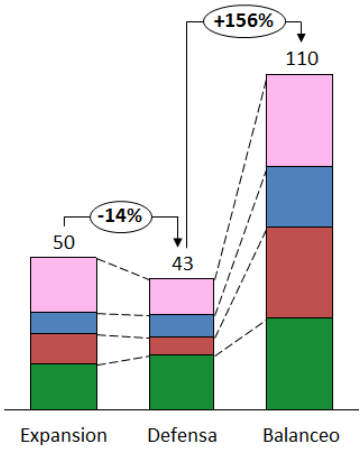


Figura 58: Evolución victorias Mapa 2.

A continuación se muestra el estado final del combate para el mapa 2, la heurística GR/C y contrincante ProspectorBot. La figura de la izquierda muestra el estado final del encuentro para el agente ofensivo-defensivo y la de la derecha, para el agente con balanceo de tropas:

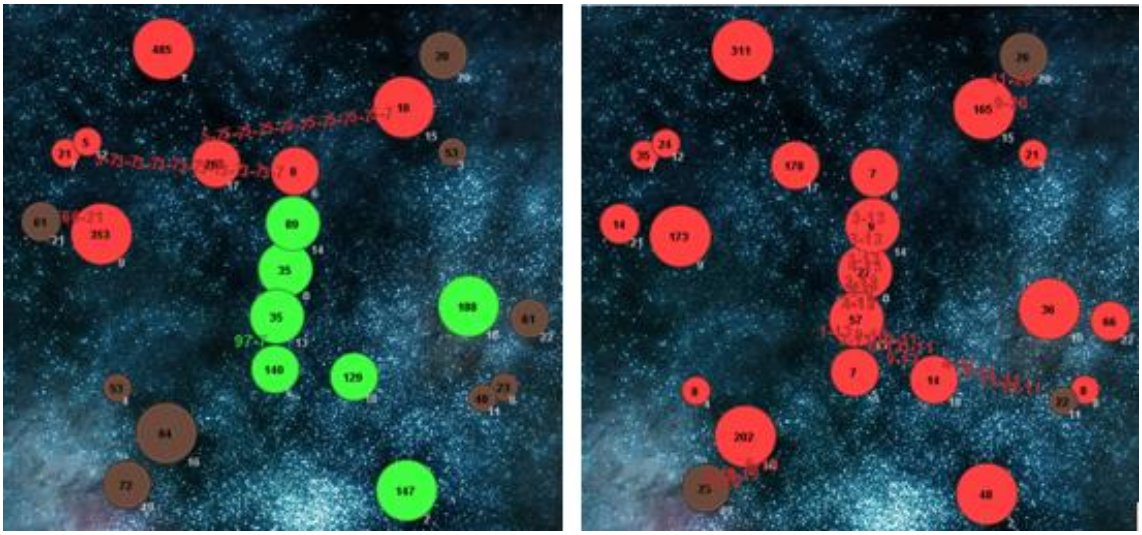


Figura 59: Escenarios de partida en el mapa 2.

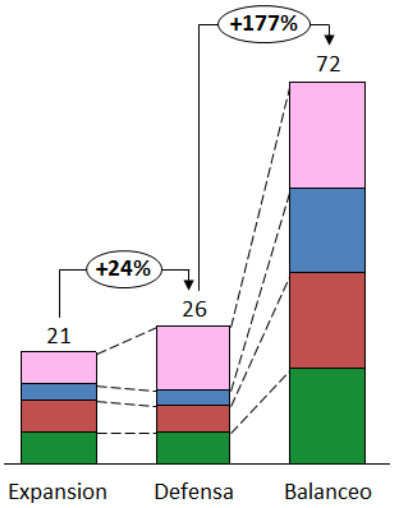


Figura 60: Evolución victorias Mapa 10.

En este caso la mejora paulatina se nota tanto en el agente defensivo como en el de balanceo de tropas. Se muestran las imágenes finales del combate con la heurística GR/(D*C), mapa 10 y contrincante ProspectorBot:

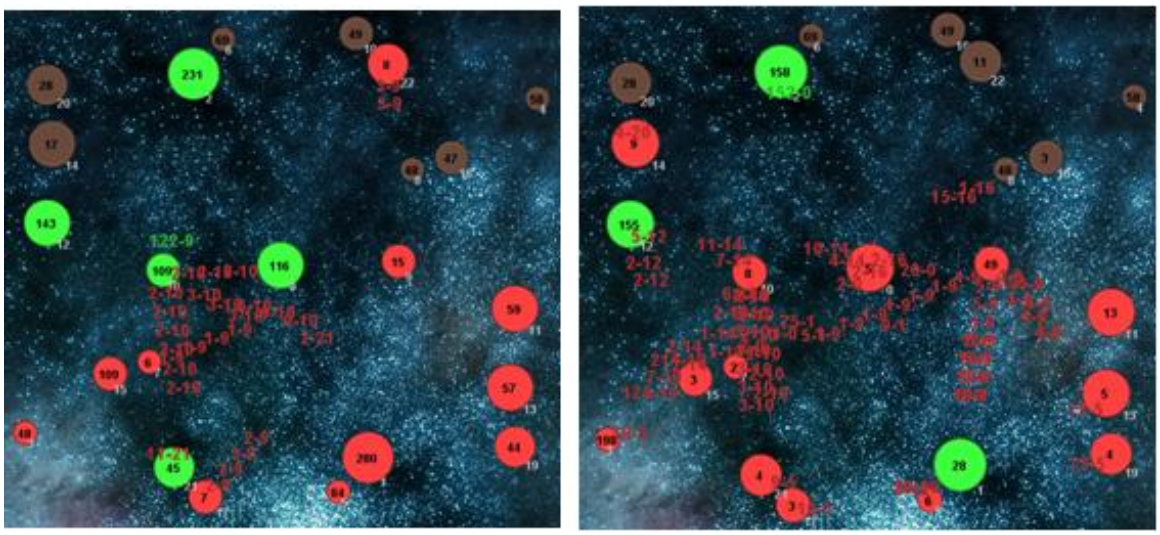


Figura 61: Escenarios de partida para el mapa 10.

5.1.4 Validación del modelo: Planet Wars.

Planet Wars es el concurso creado dentro de Google AI Challenge, organizado por la universidad del Club de Ciencias Computacionales de Waterloo y patrocinado por Google. Las bases del concurso se han explicado en el apartado 2.1 Análisis del Problema.

Tras el desarrollo del modelo expansivo, el bot se presentó al concurso, ocupando en la clasificación final la posición 1499 de entre más de 4600 participantes. El ranking final se obtuvo usando Elo ranking system (Wikipedia, 2014), un método matemático, basado en el cálculo estadístico, diseñado para medir las habilidades relativas de jugadores en juegos jugador vs jugador, como los de ajedrez o el go. La puntuación Elo de un jugador

se determina a partir de sus resultados contra otros jugadores. La diferencia de la puntuación Elo entre dos jugadores determina una probabilidad estimada de puntuación entre ellos, llamada "puntuación esperada" o expectativa. Es decir, sirve de base para predecir el desenlace de la partida. De esta manera, cuando se enfrentan dos jugadores con puntuación Elo parecida, se espera que vayan a ganar un número similar de partidas. Un jugador que tiene puntuación Elo superior que su adversario, ganará más partidas a lo largo del tiempo. Dicha puntuación se representa como un número, que se va incrementando o decrementando en función del resultado final de la partida entre dos jugadores. El ganador le quita puntos al perdedor, aumentando así su diferencia de ranking Elo.

Es interesante mencionar que, además de la clasificación personal por bots, a lo largo del concurso se recogió información adicional sobre los participantes, que permite ver estadísticas como las siguientes:

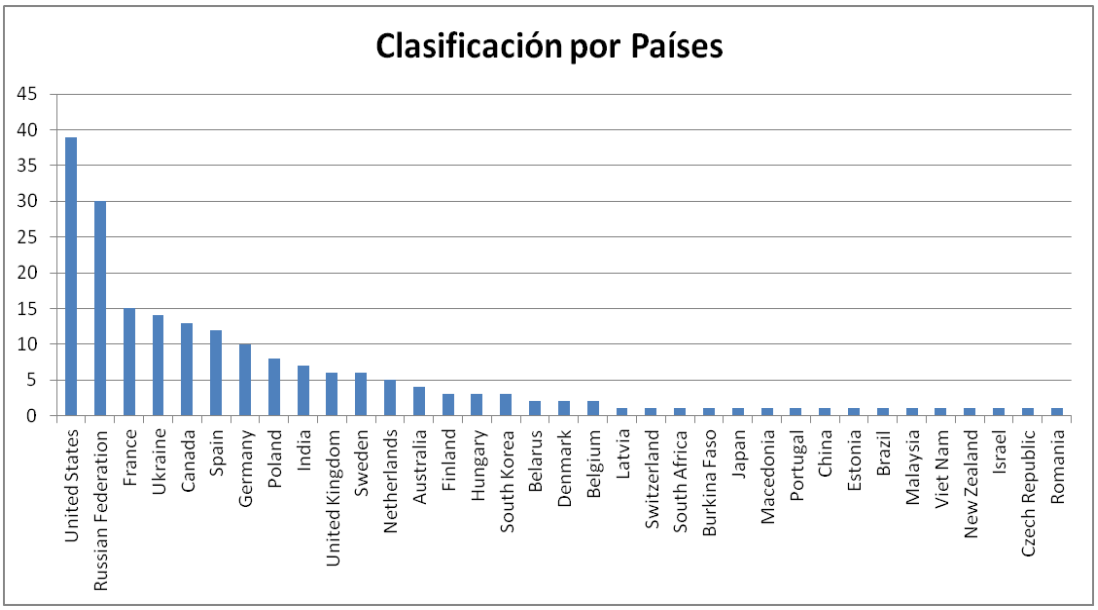


Figura 62: Clasificación por Países

Cabe destacar que España quedó en sexto lugar, por delante de países como Japón, China, Reino Unido o Suecia.

Haciendo foco en los lenguajes de programación utilizados, el más popular en España fue Java.

5.1.4.1 Primera ronda.

En cuanto al desempeño del bot desarrollado, se han realizado dos entregas. La primera consistía **únicamente en el modelo expansivo**. Como se ha mencionado anteriormente, este agente ocupó el puesto 1499 en el ranking general y el puesto 90 entre los participantes españoles. Se presentó al concurso de Google AI Challenge con el nombre Emgallar.

Para ilustrar el funcionamiento de las tácticas implementadas, se van a examinar algunos de los vídeos de los combates finales que se desarrollaron en el concurso:

Combate 1 (http://planetwars.aichallenge.org/visualizer.php?game_id=9436783)

Claramente, nuestro bot obtiene la ventaja con el primer movimiento ofensivo, en el que intenta conquistar los tres planetas más cercanos, y lo consigue en el turno 11. Eso le da una importante ventaja al generar muchas más tropas para enfrentarse en las batallas por los planetas recién conquistados contra el enemigo.

Por el contrario, algunos fallos más obvios están relacionados con no proteger sus planetas y seguir conquistando aún estando en peligro, como por ejemplo en el turno 15, cuando el contrincante acaba de arrebatar un planeta muy cercano.

En este caso, el combate se ha resuelto rápidamente, ya que ambos jugadores han comenzado en zonas adyacentes del mapa lo que inevitablemente acelera el enfrentamiento, pudiendo ganar sin necesidad de conquistar gran parte del mapa.

Combate 2 (http://planetwars.aichallenge.org/visualizer.php?game_id=9433545)

En este ejemplo, ambos participantes empiezan en puntos más alejados del mapa, lo que les permite dedicar algo de tiempo a la conquista de territorio más próximo.

Nuestro bot empieza a conquistar algo más lento que el adversario, consiguiendo 3 planetas en el turno 18, frente a 5 del enemigo. Sin embargo, la situación cambia radicalmente en el turno 92, cuando logra hacerse con el territorio central del mapa.

Combate 3 (http://planetwars.aichallenge.org/visualizer.php?game_id=9435240)

En esta ocasión, nuestro bot ha comenzado a conquistar los planetas neutrales en el mismo orden que su contrincante, hasta el turno 12, donde empieza a centrar sus esfuerzos en planetas muy costosos y poco productivos, lo cual lo coloca en una desventaja numérica.

Combate 4 (http://planetwars.aichallenge.org/visualizer.php?game_id=9445774)

En este combate nuestro bot comienza la expansión de manera similar que su enemigo, hasta el turno 45 aproximadamente. Sin embargo, los siguientes movimientos han dejado desprotegido su planeta más estratégicamente destacable, por lo que pierde toda la zona central del mapa. Aquí se ilustra una situación similar a la de la segunda partida de ejemplo, salvo que en este caso con los papeles intercambiados.

En conclusión, la expansión del bot desarrollado es bastante efectiva, conquistando planetas estratégica y numéricamente ventajosas. El principal inconveniente del bot es su falta de movimientos defensivos, que deja expuestos los planetas claves en el buen desenlace de la partida. Este fue el punto de partida para desarrollar las versiones siguientes incluyendo un modelo de defensa.

Como una mejora futura cabría destacar, la implementación de una estrategia que permitiese anticiparse a los movimientos enemigos, para poder planificar los movimientos a largo plazo.

Hay más ejemplos disponibles en la web de Planet Wars (http://planetwars.aichallenge.org/profile.php?user_id=9813).

5.1.4.2 Ronda Final

Para mejorar el resultado conseguido se ha seguido desarrollando al agente, dando lugar a la implementación de las tácticas de defensa y balanceo de tropas. Tras las primeras pruebas con el conjunto de bots de entrenamiento, cuyos resultados se explican en los apartados 4.1.2 Validación del Modelo: Defensa y 4.1.3 Validación del Modelo: Balanceo de Tropas, se ha procedido a probar el bot completo contra enemigos reales. Lamentablemente, no ha sido posible enviar al agente mejorado al concurso, ya que los plazos del mismo se habían cerrado. Por lo tanto, se ha procedido a simular los combates con aquellos participantes de Planet Wars que habían publicado su código en los foros de Galcon Ai Challenge. Los contrincantes que se han encontrado son:

Nombre Bot	Ranking Final	Lenguaje
ZerlingRush	8	Java
Dmg111	12	Python
Davidjliu	13	Python
SmlohBot	19	Java
Neverstu	29	C++
Manwe56	32	Java
Amstan	36	C++
Barabanus	41	C++
Mogron	46	C++
Deccan	47	Java
RebelXT	56	Python
Fglider	61	C++
Myrrayr	68	C++
Mistmanovx	77	C++
Error323	78	C++
Flagcapper	91	C++
Exaide	112	Java
Ludakris	122	Java
Alocaly	133	Python
Eburnette	149	Java
Krokokrusa	350	Java
Narnach	383	Ruby
InvaderZim	396	Java
Crybaby	602	PHP
Lakeweb	638	C++

Tabla 3: Ranking Bots Google AI challenge

Además, para rankear a los bots se ha utilizado un algoritmo similar al empleado en el concurso real, llamado BayesElo y publicado por Remi Coulom, profesor asociado de la universidad Université Lille 3. El tema principal de sus investigaciones es la inteligencia artificial aplicada en juegos, sobre todo el ajedrez y el go (Coulom, 2002). La

herramienta BayesElo lee los resultados de las partidas en formato PGN y estima la puntuación Elo de cada jugador basándose en una distribución de probabilidad previa y calculando una distribución posterior en función de los resultados observados.

Como se puede observar, el nivel de los jugadores enemigos es mucho mayor en esta ronda de pruebas que en todas las anteriores, estando todo ellos entre los 1000 primeros y la mayoría de ellos entre los 100 primeros del Google AI Challenge. En esta serie de pruebas se ha nombrado al agente AntBot, en honor a los algoritmos utilizados para su desarrollo.

En una primera prueba se han ejecutado 90 combates en total entre todos los jugadores, determinando a los contrincantes al azar. El resultado es el siguiente:

Rank	Nombre Bot	Elo Rating
1	SmlohBot.jar	324
2	Neverstu	299
3	RebelXT	161
4	Barabanus	131
5	Mogron	124
6	ZerlingRush	89
7	Davidjliu	85
8	iouri	79
9	Ludakris	69
10	Error323	62
11	**AntBot**	**60**
12	Manwe56	46
13	dmg111	32
14	Deccan	4
15	Exaide	-4
16	Fglider	-15
17	CryBaby	-21
18	Mistmanovx	-22
19	Eburnette	-34
20	Sspeed	-53
21	Flagcapper	-65
22	Alocaly	-74
23	Toolbot	-78
24	Krokokrusa	-106
25	Seer	-130
26	Areabot	-154
27	Murrayr	-184
28	InvaderZim	-195
29	Amstan	-212
30	Sniperbot	-216

Tabla 4: Ranking AntBot en primera ronda.

Nuestro bot se ha posicionado en la undécima posición, con una puntuación Elo de 60.

Para ampliar la muestra y asegurarnos de que es lo suficientemente representativa de la realidad, en una segunda fase se han ejecutado un total de 1860 combates entre todos los jugadores (aprox. 60 por bot), siendo el resultado como sigue:

Rank	Nombre Bot	Elo Rating
1	ZerlingRush	460
2	iouri	394
3	dmg111	382
4	Neverstu	327
5	Davidjliu	262
6	SmlohBot	255
7	Mogron	195
8	Amstan	183
9	Manwe56	174
10	Deccan.jar	153
11	Fglider	153
12	RebelXT	145
13	Error323	129
14	Murrayr	122
15	Flagcapper	108
16	Eburnette	104
17	Mistmanovx	83
18	Exaide.jar	65
19	Alocaly	48
20	Ludakris	37
21	InvaderZim	-49
22	Barabanus	-162
23	Krokokrusa.jar	-199
24	CryBaby	-258
25	Sniperbot	-277
26	Grower	-312
27	Speed	-384
28	**AntBot**	**-465**
29	Toolbot	-497
30	Seer	-564
31	Areabot	-611

Tabla 5: Ranking AntBot en segunda ronda.

Se aprecia que AntBot ha descendido a la 28-a posición, con una puntuación Elo final de -465. No es una mala posición ya que ha conseguido quedar por delante de 3

oponentes en la segunda ronda (y eso implica a más de 4000 oponentes en el concurso real) y ha conseguido ganar algunos combates que no ha podido ganar en el Google AI Challenge, antes de estar desarrollado por completo.

Con el fin de analizar el comportamiento del agente desarrollado, a continuación se muestra más detalle sobre los combates perdidos y ganados por enemigo y mapa:

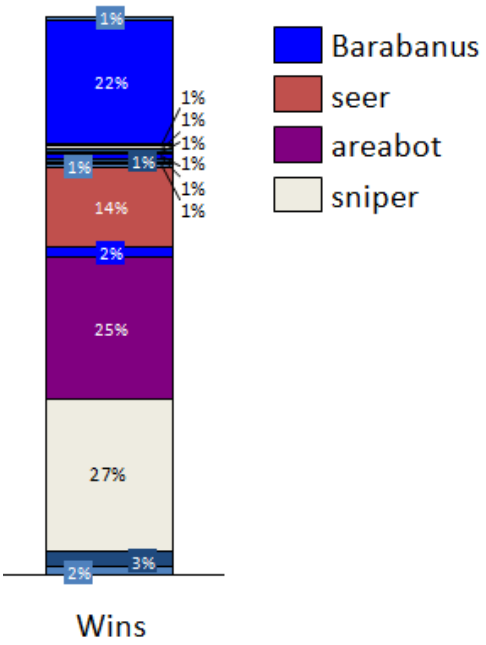


Figura 63: Victorias de AntBot en la Segunda ronda

En la anterior gráfica se analiza el detalle de los combates ganados por AntBot. Se puede observar que hay cuatro enemigos ante los que AntBot claramente tiene algún tipo de ventaja, ya que les vence más que a los demás: Barabanus, Seer, Areabot y Sniper.

Para hacer un análisis más cualitativo, estudiamos algunos de los combates en los que participen esos jugadores.

Areabot. Mapa 1_45

En este combate la ventaja principal se consigue con una expansión inicial más agresiva que la del enemigo. Mientras Areabot conquista sólo un planeta en el primer turno, AntBot inicia el movimiento para conquistar cuatro, lo cual supone una gran ventaja en término de tropas al inicio del combate. En el turno 5 AntBot ha terminado de conquistar los primeros tres planetas, y en el turno 7 conquista otro más, generando así 13 y 18 tropas más por turno que su contrincante.

Seer. Mapa 1_18.

En este caso la rápida expansión inicial ayuda a AntBot a acumular tropas rápidamente al inicio del combate y le da ventaja a lo hora de sus futuras conquistas, al igual que ocurre en el combate anterior. En el primer turno AntBot inicia la conquista de 3 planetas, frente a 2 del enemigo. En el turno 6 ambos jugadores tienen el mismo número de planetas conquistados, pero en el turno 8 llegan las 11 tropas de AntBot al tercer planeta, adelantando al contrincante en productividad. En el turno 26 AntBot ya lleva 2 planetas de ventaja, por lo que la victoria se hace inminente.

Fglider. Mapa 2_28

En este combate la ventaja principal consiste en la ocupación de los planetas centrales, que permite ganar ventaja estratégica a la hora de atacar al enemigo o defender los planetas propios. En el turno 16 los dos jugadores van bastante igualados, incluso Fglider ha conquistado 1 planeta más, aunque de baja producción. En el turno 47 AntBot conquista 3 de los 5 planetas restantes, lo cual cambia drásticamente el panorama. A continuación, conquista el resto de los planetas centrales, incluso arrebatándole uno al oponente, pasando a dominar la batalla. El contrincante intenta reconquistar, sin éxito los planetas centrales, o bien atacar planetas periféricos. Sin embargo, como AntBot está posicionado en el centro del mapa, está más cerca de sus planetas periféricos que Fglider, que se ve obligado a atacar desde una esquina del mapa. Eso le permite defenderse a tiempo, ya que sus tropas tardan menos en llegar, como se puede observar en el turno 96.

InvaderZim. Mapa 1_32

En este combate ocurre algo similar al anterior. AntBot ocupa uno de los planetas centrales en el turno 30, cerrando así el paso al jugador contrario. Éste no ataca los planetas adyacentes y se centra en acumular tropas, pero al final del combate AntBot gana por número de tropas, ya que ha conquistado más planetas que el enemigo.

Sniperbot. Mapa 2_29.

Este caso agrupa las ventajas anteriormente descritas. AntBot conquista de manera rápida y eficiente 4 planetas en los primeros 7 turnos, uno de los cuales es el planeta central del mapa. En el turno 10 conquista los 3 planetas centrales, arrebatando uno a Sniperbot y asegurándose no sólo las tropas sino también una buena posición geográfica para futuros movimientos.

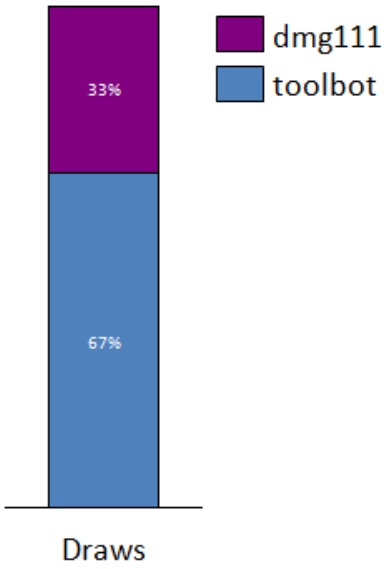


Figura 64: Empates de AntBot en la segunda ronda.

Esta gráfica muestra los combates empatados. Se aprecia que AntBot empata únicamente con dos bots: dmg111 y toolbot. Más bien suele ganar o perder los combates contra sus contrincantes, por lo que la muestra es demasiado pequeña para poder sacar conclusiones más detalladas.

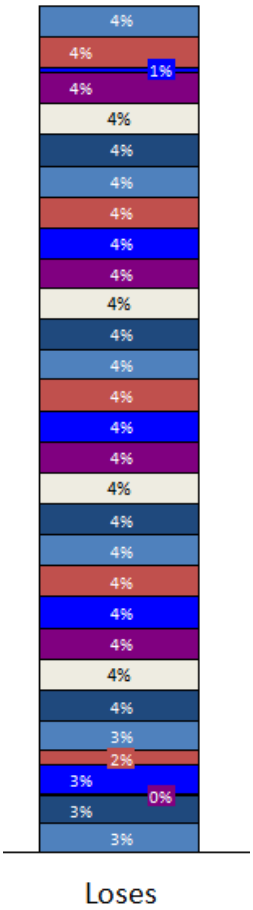


Figura 65: Derrotas de AntBot en la segunda ronda.

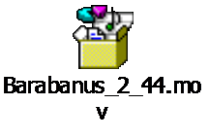
En cuanto a los combates perdidos, que se muestran en la figura anterior, aparece una mezcla muy heterogénea de rivales, todos con un porcentaje bastante similar. Eso indica que no hay un contrincante cuya estrategia de combate sea especialmente efectiva.

A continuación se muestran algunos ejemplos de batallas:

Amstan. Mapa 1_46.

En este caso, AntBot se ha expandido hacia los planetas más cercanos primero, mientras que el contrincante ha conquistado más rápidamente los planetas centrales. En el turno 14 ambos jugadores tenían 6 planetas cada uno. La diferencia radica en que dos de los planetas conquistados por Amstan son planetas centrales, lo cual le ha permitido ganar ventaja y establecer una zona "segura". A partir de ahí, ambos han acumulado tropas, pero al tener más planetas conquistados, el enemigo se ha llevado la victoria. Podría decirse que es el caso contrario al combate contra InvaderZim en el mapa 1_32, pero siendo AntBot el más lento de los dos.

Barabanus. Mapa 2_44.



Este combate parece fácil de ganar al principio ya que el contrincante no realiza ningún movimiento y acumula tropas hasta el turno 53. Sin embargo, en ese momento, el enemigo ataca rápidamente, conquistando 4 planetas más cercanos en el turno 59. El principal fallo de AntBot es el hecho de haber dejado descubiertos los planetas cercanos al enemigo. Además, AntBot tarda en cambiar la dirección del envío de tropas; para cuando logra reaccionar ya es demasiado tarde y sus primeras tropas de defensa llegan 1 turno después de que los planetas hayan sido conquistados.

Iouri. Mapa 2_11.

El enemigo en este caso es el tercer finalista del concurso. El combate se resuelve rápidamente ya que el oponente tiene un ataque muy agresivo y conquista enseguida los planetas más cercanos. En el turno 8 conquista 3 planetas frente a 2 que conquista AntBot. A continuación se expande hasta ocupar 5 planetas en el turno 18, mientras AntBot tiene 3. En los siguientes turnos Iouri acumula tropas y elimina a AntBot conquistando todos sus planetas.

Para poder buscar pistas de principales oportunidades de mejora, se enfocará el análisis de derrotas usando el segundo factor que interviene en el combate: los mapas.

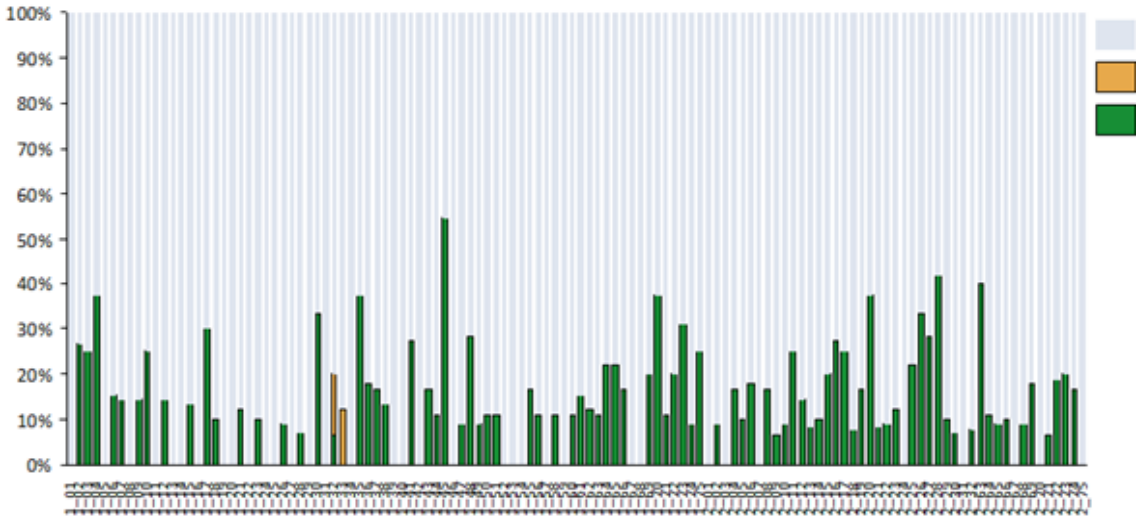


Figura 66: Detalle por mapa de victorias de AntBot en segunda ronda.

En la Figura 66 se pueden distinguir algunos mapas en los que el porcentaje de victorias es más elevado que el resto:

1_45

1_73

2_36

2_28

1_10

1_02

2_54

1_75

Tabla 6: Mapas con mayor % de victorias de AntBot en segunda ronda.

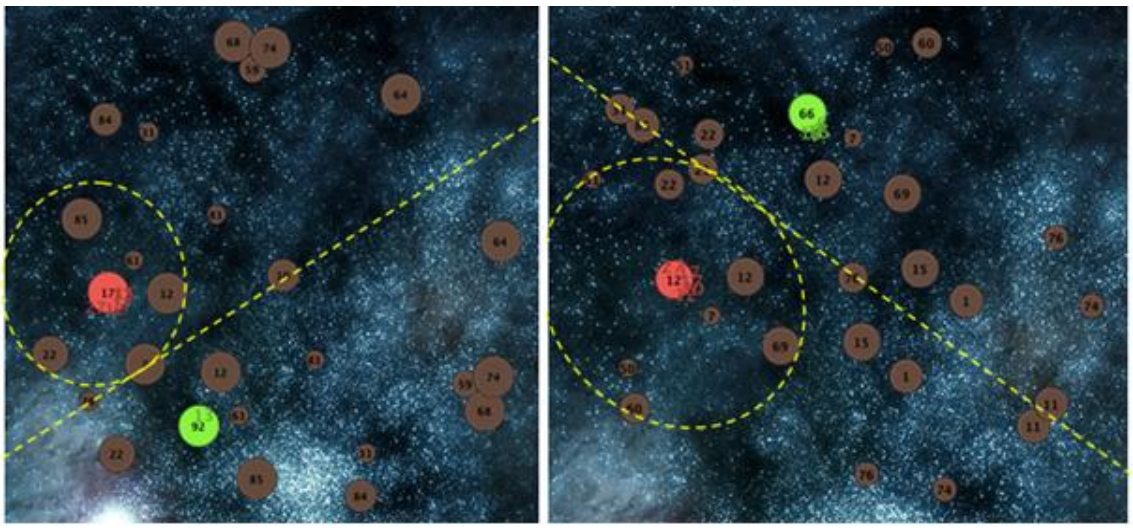


Figura 67: Mapa 1_45 (izda.) y Mapa 1_73 (dcha.).

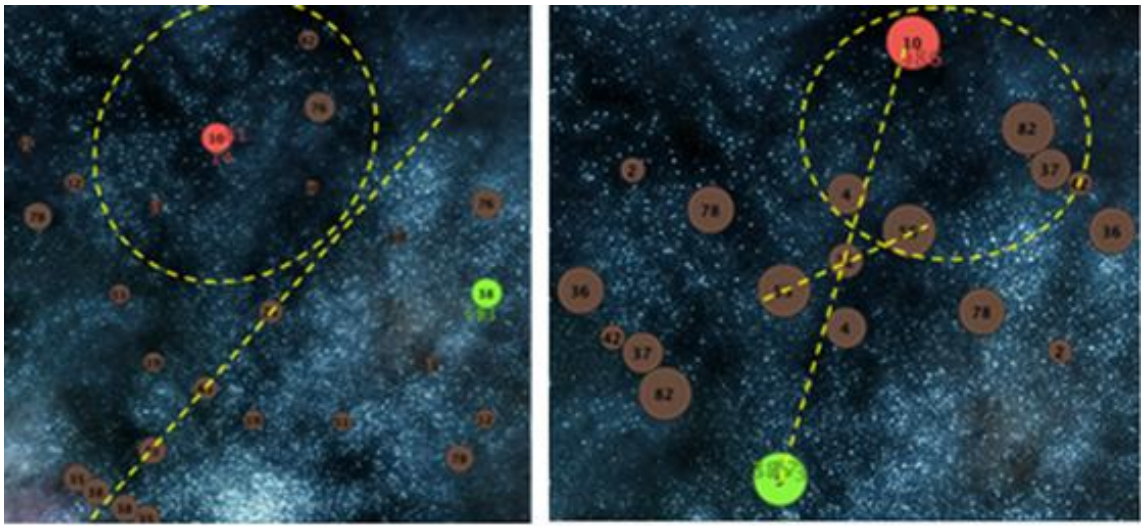


Figura 68: Mapa 2_36 (izda.) y Mapa 2_38 (dcha.)

También se puede distinguir un conjunto de mapas en los que el bot no gana ninguna batalla:

2_13
1_39
1_19
1_10
1_22
2_09
2_71

Tabla 7: Mapas con menor % de victorias de AntBot en segunda ronda.

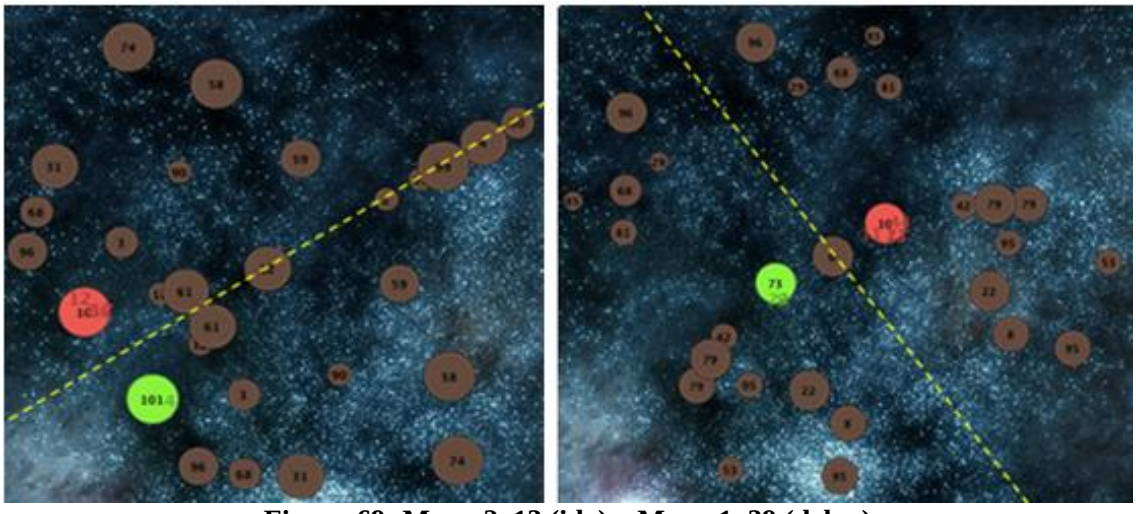


Figura 69: Mapa 2_13 (ida) y Mapa 1_39 (dcha.)

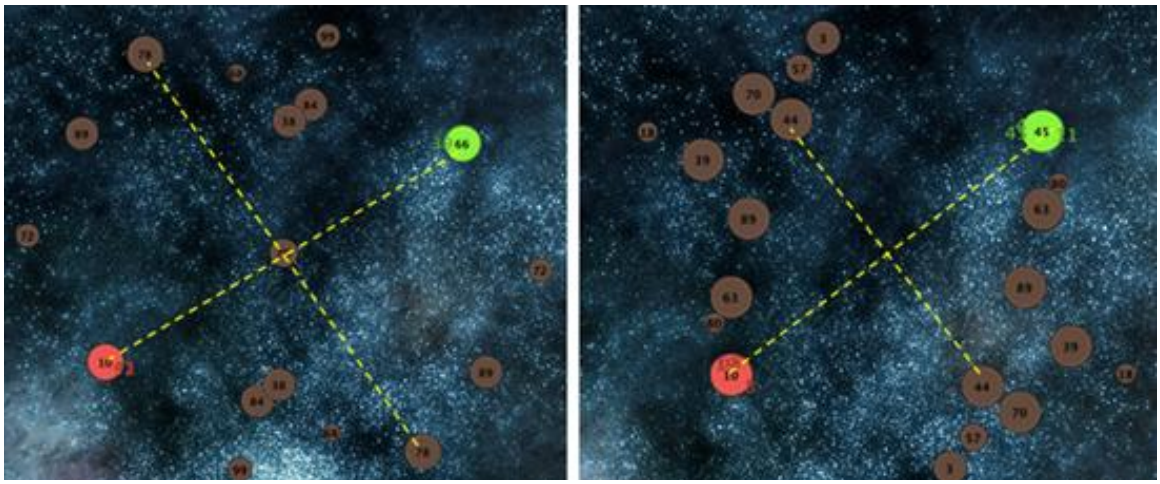


Figura 70: Mapa 1_19 (izda.) y Mapa 1_10 (dcha.)

En base a estos ejemplos se puede observar que el primer conjunto de mapas (Fig. 88 y 89), donde el número de victorias es mayor, posee una distribución de planetas con una característica similar: hay una serie de planetas neutrales entre los dos planetas iniciales, al comienzo del combate. Hay algunos mapas en los que los que esos planetas neutrales no están exactamente entre ambos jugadores pero sí quedan más cerca del planeta inicial de cada uno que el de su oponente. Esta situación de partida permite que ambos jugadores centren sus esfuerzos en la conquista de esos planetas neutrales, acumulando un cierto número de tropas antes de pasar al ataque. Como hemos visto en las partidas de ejemplo, AntBot normalmente tiene mejor desempeño en la conquista que en la defensa, por lo que poder acumular tropas al inicio del juego le da una ventaja numérica que ayuda a paliar los fallos de defensa que pueda cometer.

El segundo conjunto de mapas, con mayor número de combates perdidos, presenta una distribución geográfica inversa: Hay muy pocos o ningún planeta directamente entre los dos planetas iniciales, por lo que se ven obligados a enzarzarse en combate mucho antes. Eso elimina el colchón de tropas que puede acumular AntBot para protegerse de los posibles ataques enemigos.

Por último, en naranja están señalados los dos mapas (Mapa 1_32 y mapa 1_33) en los que se han dado combates empatados.

Capítulo 6

Conclusiones y líneas futuras

En este capítulos se describen las conclusiones que se han sacado como fruto del trabajo realizado, así como se esbozan las líneas futuras que se podrían seguir para profundizar en este proyecto.

6.1 Conclusiones

Este proyecto se inició con dos principales objetivos en mente:

- Aplicar los conocimientos adquiridos a lo largo de la carrera en un entorno real y competitivo, aprovechando la oportunidad que presentaba el concurso de Google AI Challenge. Desde este punto de vista, la realización de este trabajo ha contribuido sin duda a afianzar los conceptos teóricos aprendidos y ampliarlos por cuenta propia como parte de la investigación realizada, tanto para la implementación y sus sucesivas mejoras, como para la posterior documentación del proyecto.
- Modelar y desarrollar un agente que resuelva un problema complejo presentado en forma de un videojuego. La fase de modelado ha supuesto un gran reto, dada la complejidad del problema. Se han pensado y probado múltiples modelos de representación del universo del propio juego y de parametrización de la función objetivo. Por ejemplo, estaba relativamente claro que los planetas a conquistar se presentarían en forma de un grafo, con todos los nodos conectados, sin embargo

ha costado refinar la información que caracterizaría a cada planeta o cómo decidir cuál de todas las características tomaría mayor o menor relevancia, dependiendo del objetivo inmediato que se intenta conseguir: conquistar, defender o redistribuir las tropas. Tener un adversario contra el que competir también ha aumentado la dificultad del problema, ya que ha introducido otro factor variable en el universo que había que tener en cuenta. Se han barajado y, en la mayoría de las veces, implementado, probado y desechado, varias aproximaciones, desde las más reactivas y defensivas, hasta aquellas que intentaban aprender las estrategias del contrincante para anticiparse a ellas. Finalmente, se ha diseñado una solución que busca el equilibrio entre el esfuerzo invertido y los resultados obtenidos, que es la que ha sido expuesta a lo largo de este documento.

Una vez en la fase de pruebas, los resultados han validado el modelo planteado en gran medida. Las pruebas iniciales han confirmado el potencial del modelo, sobre todo en la fase de conquista, donde se adapta a las características de los distintos mapas y comienza la expansión de forma ventajosa, sobre todo en escenarios donde los jugadores empiezan lejos el uno del otro. Sin embargo, la fase de combate ha sido bastante más compleja, tanto el diseño, que se ha comentado anteriormente, como la implementación, que ha supuesto el desarrollo de un sistema nuevo de defensa. Como consecuencia, se ha visto una mejoría de los resultados, aunque con menor impacto del esperado. Es en este punto donde se ha hecho una exhaustiva labor de análisis de los puntos fuertes y las principales oportunidades de mejora que presenta el agente implementado.

Entre los puntos fuertes está la adaptabilidad del modelo, tanto la real como la potencial. Por adaptabilidad real se entiende la capacidad del agente a evaluar el estado de la partida y rutas alternativas en todo momento, puesto que la situación cambia cada turno. Por adaptabilidad potencial se entiende la capacidad de acomodarse a cambios en las especificaciones del juego, siempre y cuando no sean tan radicales como para cambiar sus reglas por completo. Por ejemplo, es posible modificar o ampliar las características de los planetas, variar la importancia de cada una de esas características para ciertas decisiones, o incluso el propio escenario del juego, siempre y cuando pueda representarse en forma de un grafo. También es posible incluir nuevas reglas de comportamiento con relativa facilidad gracias a la modularidad del código que soporta el modelo, si bien la implementación inicial no es tan inmediata como podría ser la de un diseño específico.

Otra de las ventajas importantes es la escalabilidad del modelo. Aunque en este proyecto no se han hecho pruebas específicas con mapas más grandes, por quedar fuera del alcance del proyecto ya que no estaba permitido en el juego inicial, en un caso hipotético de que el mapa se ampliara, la solución implementada seguiría siendo válida y no necesitaría desarrollos adicionales. Esta ventaja adquiere más peso si se toma en cuenta que el marco para este proyecto es un videojuego, donde la configuración y personalización juega un papel decisivo en la experiencia del cliente.

Relacionado con este último punto, está la capacidad para parametrizar el desempeño del agente, para que sea más o menos eficiente en la búsqueda de soluciones, simplemente cambiando las variables inherentes al modelo, como la evaporación de la feromona o la probabilidad de escoger la mejor solución en cada paso.

En cambio, una de las principales áreas de mejora es el modelo defensivo y la gestión del combate. Si bien las pruebas han mostrado que tanto la defensa como el balanceo de tropas han ayudado a mejorar los resultados, no ha sido lo suficiente como para asegurar la victoria en la mayoría de los casos. Los peores escenarios han sido con los oponentes más agresivos o donde el combate empezaba pronto, porque los jugadores comenzaban en puntos cercanos del mapa.

Otro de los inconvenientes de cualquier algoritmo genérico es que son normalmente menos eficientes que otros desarrollados específicamente para el problema en cuestión. En este caso, no se han hecho pruebas exhaustivas para medir el tiempo promedio por turno de cada jugador, pero sí se ha llegado al límite de 1 segundo por turno establecido por el concurso, por lo que se han tenido que re-evaluar y re-implementar algunas estrategias. No obstante, para escenarios con mapas más amplios puede resultar más eficiente que un algoritmo específico, ya que el coste de la búsqueda exhaustiva se eleva exponencialmente.

Por todo ello, si bien no se ha ganado el concurso, el modelo implementado presenta múltiples ventajas frente a otros concursantes, sobre todo evaluándolo en el marco más amplio de factores como la escalabilidad y la adaptabilidad.

Por último, comentar que este proyecto ha requerido un gran esfuerzo de desarrollo adicional a lo que es únicamente el agente. Dada la cantidad de información a evaluar, se ha tenido que modificar la interfaz inicial proporcionada por los organizadores del concurso para reflejarlo en los logs y poder estudiarla posteriormente. Además, se ha implementado una interfaz gráfica adicional, para visualizar las partidas, añadiendo información extra relativa al modelo, como la cantidad de feromona en los caminos, o la deseabilidad de los planetas, para analizar y testear el funcionamiento del agente. Finalmente, se ha desarrollado un módulo para la automatización de las baterías de pruebas y registro de resultados en formato específico requerido para el rankeo de Bayeselo en la simulación del concurso en las pruebas finales.

6.2 Líneas futuras.

Como se ha mencionado en el apartado anterior, hay muchos aspectos interesantes de este proyecto que no se han podido implementar o desarrollar con la profundidad deseada, por quedarse fuera del ámbito de este proyecto que se centra en aplicar un algoritmo ACO a un escenario concreto. Algunos de esos aspectos son:

- Configuración de mapas. Los mapas y la distribución de los planetas son uno de los elementos fundamentales del escenario de juego y los más sencillos de modificar. Desde el punto de vista de los videojuegos, aporta el grado de personalización necesario para que la partida sea diferente cada vez y el juego no aburra. Desde el punto de vista de la aplicación de algoritmos ACO, sería muy interesante generar y hacer pruebas exhaustivas del comportamiento del agente desarrollado en mapas con distintas características, como mayor número de planetas o eliminando la simetría. Sería muy interesante medir su rendimiento y comprobar su adaptabilidad, comparándolo con los algoritmos de otros concursantes.
- Opciones multijugador. El juego admite la posibilidad de jugar con más de dos jugadores, con todo un abanico de posibles estrategias. Una de las líneas futuras consistiría en explorar el comportamiento del agente en esos escenarios, y cómo se adapta al reto de evaluar acciones de más de un enemigo, lo cual introduciría variables adicionales a considerar. Dependiendo de la inteligencia de los adversarios, se podrían producir alianzas o simplemente aprovechar el oportunismo y atacar a un jugador que ya esté siendo atacado por otros.

- Optimización de rendimiento. Otra de las posibles líneas futuras a explorar es el rendimiento del algoritmo en términos del tiempo que se tarda en realizar cada turno. No se ha realizado en este proyecto por no haber un registro del tiempo de cada turno por defecto, por lo que se requeriría desarrollo adicional para llevarlo a cabo. Sin embargo, sería muy interesante comparar el rendimiento del modelo, tanto en el escenario inicial como en los comentados en los puntos anteriores, con mapas más amplios o varios contrincantes.
- Otro de los aspectos muy interesantes es el de disponer de otro jugador contra el que competir por los recursos en la partida. Sólo por el mero hecho de su existencia, ofrece todo un abanico de posibilidades en cuanto a las interacciones entre ambos jugadores y estrategias a seguir con el fin de conseguir la victoria. Sería muy interesante combinar algoritmos ACO con otras técnicas de la IA para diseñar estrategias más avanzadas, que permitan, por ejemplo, anticiparse a los movimientos del adversario o aprender de las partidas anteriores.

Por último, destacar que ha sido un trabajo retador, a la par que muy interesante y motivador, gracias al auge de la Inteligencia Artificial en los últimos tiempos y su aplicabilidad en múltiples ámbitos de la vida cotidiana. Además, la temática relacionada con los videojuegos y la componente competitiva del concurso han contribuido a mantener dicho interés y motivación. Como resultado de este trabajo he tenido la oportunidad de afianzar los conocimientos teóricos adquiridos, así como experimentar las ventajas e inconvenientes de aplicar ACO en modelos reales, tales como la escalabilidad y la adaptabilidad o, en cambio, una mayor dificultad de diseño y necesidad de muchas pruebas ya que los resultados no son fácilmente previsibles de forma teórica. Además de cumplir el objetivo inmediato de resolver la partida, he tenido la oportunidad de evaluar la IA como una componente más dentro del marco de un videojuego. Bajo este enfoque han cobrado mayor relevancia factores nuevos, distintos a los meramente técnicos de rendimiento y optimización, como la variabilidad de las partidas, el grado de personalización del agente o el control de la dificultad, que la IA es capaz de satisfacer y que resultan fundamentales en la experiencia del cliente final, lo cual a su vez ha sido clave en la expansión de una industria estratégica, como la de los videojuegos.

Glosario

ACO	*Ant Colony Optimization.* *Es una familia de algoritmos derivados del trabajo realizado por Dorigo et al., (Dorigo, Maniezzo, & Colorni, 1991), basada en el comportamiento social de las hormigas, las cuales usan una forma de comunicación basada en sustancias químicas denominadas feromona.*
A-life	*Artificial life.* *En este trabajo se refiere a un sub-género de videojuegos de simulación en los que el jugador controla una o más formas de vida artificial.*
Bot	*Aplicación software que ejecuta tareas sencillas y repetitivas. En este trabajo se ha referido así a los agentes programados por los concursantes del Google AI Challenge.*
EC	*Computación Evolutiva.* *Comprende paradigmas de clasificación y optimización que se apoyan en la combinación de las teorías de la evolución y teorías de la computación.*
Elo Ranking System	*Es un método matemático, basado en el cálculo estadístico, diseñado para medir las habilidades relativas de jugadores en juegos jugador vs jugador, como los de ajedrez o el go.*
Fitness	*Función que determina la calidad de la solución dentro de un problema de optimización de colonias de hormigas.*

FPS	*First Person Shooter (Juego de Disparos en Primera Persona).*
GA (AG)	*Genetic Algorythms (Algoritmos Genéticos).* *Son un sub-género de algoritmos evolutivos, que se basan el concepto de evolución biológica de Darwin.*
GR	*Growth Rate.* *Es la tasa de producción de tropas por turno de cada planeta en Planet Wars.*
IP	*Índice de Peligrosidad.* *Es el número de planetas enemigos de los que un planeta aliado está más cerca que ningún otro planeta aliado.*
MKP	*Multidimensional Knapsack Problem*
MMAS	*Max – MIN Ant System*
Pathfinding	*En el contexto de videojuegos hace referencia a la capacidad del programa de encontrar el camino más corto entre dos puntos cualesquiera del mapa y llegar a su destino.*
PGN	*Portable Game Notation.* *Es un formato para grabar partidas de ajedrez, que incluye información de los jugadores, movimientos, resultado, etc.*
PNJ (NPC)	*Personaje No Jugador (Non Player Character).* *Usado en videojuegos, es un personaje controlado por el programa y no por un humano.*
QAP	*Quadratic Assignment Problem.*
Swarm Intelligence	*Inteligencia de Enjambres de Partículas.* *Es una disciplina dentro de la Inteligencia Artificial que se ocupa de crear sistemas multi-agente inteligentes, inspirándose en el comportamiento colectivo de insectos sociales como las hormigas, abejas, termitas, u otras sociedades animales como las bandadas de pájaros o bancos de peces.*
TSP	*Travel Salesman Problem.*
UML	*Unified Modelling Language.*
VRP	*Vehicle Routing Problem.*

Bibliografía

Aarts, E., & Lenstra, J. (2003). *Local search in combinatorial optimization.* Eindhoven y Atlanta: Princeton University Press.

Asociación Española de Empresas Desarrolladoras de Videojuegos y Software de Entretenimiento. (2014). *Libro Blanco del Desarrollo español de los videojuegos.* Madrid: Autor.

Blum, C. (2004). Theoretical y Practical aspects of Ant Colony Optimization. *Dissertation in Artificial Intelligence, 282*. [s.n]: IOS Press.

Buckland, M. (2002). AI techniques for game programming. Ohio: Premier Press.

Charles, D., Fyfe, C., Livingstone, D., & McGlinchey, S. (2008). *Biologically Inspired Artificial Intelligence for Computer Games.* Londres: IGI Publishing.

Coello, C. A. (2007). *Optimización en Ingeniería.* Recuperado el 12 de junio de 2012, de Departamento de Computación de CINVESTAV: http://delta.cs.cinvestav.mx/~ccoello/optimizacion/clase1-opt-2007.pdf

Coulom, R. (2002). *Rémi Coulom's Home Page*. Recuperado el 6 de marzo de 2014, de http://remi.coulom.free.fr/

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms. *PhD Tésis, Dipartimento Di Elettronica e Informazione, Politecnico di Milano*. Milan: [s.n.].

Dorigo, M., & Di Caro, G. (1999). The Ant Colony Optimization Meta-heuristic. En D. Corne, M. Dorigo, & F. Glover, *New Ideas in Optimization* (págs. 11-32). Londres: McGraw-Hill.

Dorigo, M., & Gambardella, L. M. (1997). Ant Colonies for the Travelling Salesman Problem. *Biosystems, 43*(2), (págs. 73-81).

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization.* [s.n.]: Massachusetts Institute of Technology.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). Positive feedback as a search strategy. *Technical report 91-016*. Milan: Dipartamento Di Elettronica, Politecnico Di Milano, IT.

Fidanova, S. (2007). Probabilistic Model of Ant Colony Optimization for Multiple Knapsack Problem. *6th International Conference, Large-Scale Scientific Computing 2007* (págs. 545-552). Sozopol, Bulgaria: Springer Berlin Heidelberg.

Gambardella, L. M., & Dorigo, M. (1995). Ant-Q: A reinforcemente learning approach to the travelling salesman problem. *Proceedings of ML-95, the Twelfth International Conference on Machine Learning* (págs. 252-260). Tahoe City, California: Morgan Kaufmann Publishers.

Gartner. (2013). *Gartner Says Worldwide Video Game Market to Total $93 Billion in 2013*. Recuperado el 31 de enero de 2015, de http://www.gartner.com/newsroom/id/2614915

Grötschel, M., & Lovasz, L. (1995). *Handbook of combinatorics* (Vol. 2). Cambridge, Massachusetts: The MIT Press y North-Holland.

Guerra Marrero, A. (31 de enero de 2010). *Optimización basada en colonias de hormigas.* Recuperado el 12 de mayo de 2013, de Razón Artificial. La ciencia y el arte de crear videojuegos.: http://razonartificial.com/2010/01/optimizacion-basada-en-colonias-de-hormigas-en-la-naturaleza/

Hernandez Oliva, G. (7 de mayo de 2006). *Métodos Clásicos de Optimización para Problemas No-Lineales sin Restricciones.* Recuperado el 9 de junio de 2013, de https://www.u-cursos.cl/ingenieria/2010/2/MA3701/1/material_docente/bajar?id_material=329674

Hoffman, K. L. (2000). Combinatorial optimization: Current successes and directions for the future. *Journal of Computational and Applied Mathematics, 124*, (págs. 341-360).

Maniezzo, V., Dorigo, M., & Colorni, A. (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Transactins on Systems, Man and Cybenetics, 26*(1), (págs. 29-41).

Martin, E., Martinez, M., Recio, G., & Saez, Y. (2010). Pac-mAnt: Optimization Based on Ant Colonies Applied to Developing an Agent for Ms. Pac-Man. *IEEE Conference on Computational Intelligence and Games (CIG) 2010*, (págs. 458-464).

Miller, G. F., Todd, P. M., & Hedge, S. U. (1989). Designing neural networks using genetic algorithms. En J. D. Schaffer, *Procedings of the Third International Conference on Genetic Algorithms* (págs. 379-384). San Francisco: Morgan Kaufmann Publishers.

Muñoz, M., López, J., & Caicedo, E. (2008). Inteligencia de enjambres: sociedades para la solución de problemas (una revisón). *Ingeniería e Investigación, 28*(2), (págs. 119-130).

Navas, J. A. (2014). *Radiografía de la industria del videojuego en España en 9 cifras*. Recuperado el 2 de febrero de 2015, de Elmundo.es: http://www.elmundo.es/tecnologia/2014/05/22/537e1585ca47417b0d8b457e.html

Optimización (matemática). (12 de junio de 2012). Recuperado el 10 de junio de 2013, de Wikipedia: http://es.wikipedia.org/wiki/Optimizaci%C3%B3n_(matem%C3%A1tica)

Optimización combinatoria. (10 de enero de 2012). Recuperado el 15 de junio de 2012, de Wikipedia: http://es.wikipedia.org/wiki/Optimizaci%C3%B3n_combinatoria

Osman, I. H. (1996). *Meta-Heuristics: theory and applications.* Boston: Kluwer Academic.

Raposo, M. (2008). Estructura y evolución reciente de la industria del videojuego. *Palermo Business Review, 1*, (págs. 61-72).

Sait, S. M. (1999). *Iterative computer algorithms with applications in engineering: Solving combinatorial optimization problems.* Piscataway: IEEE Computer Society Press.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*(2), (págs. 99-127).

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2005). Evolving neural network agents in the NERO video game. *IEEE 2005 Symposium on Computational Intelligence and Games (CIG) 2005*, (págs. 182-189).

Wikipedia. (1 de noviembre de 2014). *Artigicial Intelligence (video game)*. Recuperado el 20 de noviembre de 2014, de Wikipedia: http://en.wikipedia.org/wiki/Artificial_intelligence_(video_games)

Wikipedia. (28 de febrero de 2014). *Elo Rating System*. Recuperado el 3 de marzo de 2014, de http://en.wikipedia.org/wiki/Elo_rating_system

Planificación

En esta sección se presenta tanto la planificación inicial que se ha hecho del proyecto como su seguimiento posterior, identificando y analizando las posibles desviaciones que se hayan producido.

Planificación Inicial

Inicialmente, el proyecto se contempló en dos fases, abarcando un período de ocho meses, desde Septiembre 2010 hasta Mayo 2011. La primera fase se compone de todas las tareas de análisis y estudio de la problemática planteada, así como del desarrollo del sistema hasta el momento de presentarlo en el concurso Google AI Challenge 2010. Las fechas del inicio del proyecto y de las primeras entregas fueron condicionadas por las fechas de publicación y de apertura y cierre del plazo de entrega del propio concurso. Posteriormente se han realizado mejoras fuera del plazo del proyecto, para complementar el trabajo inicial. Las tareas relativas a la realización de pruebas de desempeño de AntBot, diseño e implementación de mejoras, así como el análisis de los resultados se agruparon en la segunda fase del proyecto. Por último, se han incluido una serie de tareas cross relacionadas con la documentación del proyecto, a lo largo de ambas fases.

A continuación se muestra el detalle de las tareas de cada fase y el diagrama de Gantt que ilustra la planificación realizada de manera gráfica.

Nombre de la tarea	Duración	Comienzo	Fin
Fase 1: Google AI Challenge 2010	**97 días***	**01/09/2010**	**25/11/2010**
Publicación Resultados	**0 días**	**01/12/2010**	**01/12/2010**
Fase 2: Mejoras a AntBot	**91 días***	**01/12/2010**	**03/03/2011**
Documentación	**151 días***	**25/11/2010**	**20/04/2011**
Entrega	**0 días***	**02/05/2011**	**02/05/2011**

Tabla 8: Planificación inicial: resumen de tareas.

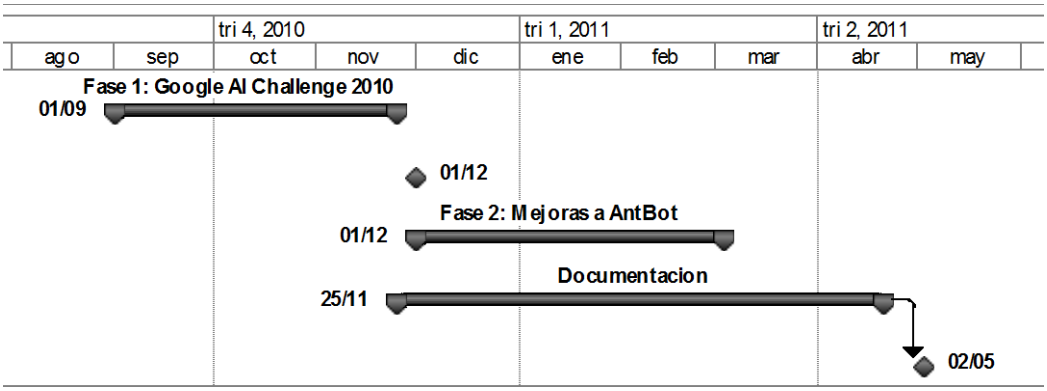


Figura 71: Planificación inicial: diagrama de Gantt completo.

Se puede observar que hay dos hitos importantes en este proyecto, siendo el primero de ellos la publicación de los resultados del concurso, que finaliza la primera fase y da comienzo a las mejoras que se van a acometer en la segunda. El segundo hito importante es la entrega final del proyecto. La duración de todas estas fases está estimada en días naturales, asumiendo una disponibilidad de lunes a sábado de 4 horas, y está marcada con símbolo '*' por ser una estimación inicial.

En las siguientes tablas y figuras se muestra el detalle de cada tarea dentro de las distintas fases:

Nombre de la tarea	Duración	Comienzo	Fin
Fase 1: Google AI Challenge 2010	**97 días***	**01/09/2010**	**25/11/2010**
Estudio Planet Wars y bases del concurso	2 horas*	01/09/2010	01/09/2010
Descarga e instalación del material	1 día*	01/09/2010	02/09/2010
Partidas de prueba Galcon manual	1 día*	02/09/2010	02/09/2010
Análisis del problema	7 días*	03/09/2010	09/09/2010
Diseño del modelo expansivo	6 días*	09/09/2010	15/09/2010
Implementación	15 días*	15/09/2010	28/09/2010
Pruebas Bots Starter Package	6 días*	28/09/2010	04/10/2010
Primera Entrega del Bot	0 días*	04/10/2010	04/10/2010
Pruebas heurísticas	10 días*	04/10/2010	12/10/2010
Implementación de mejoras	20 días*	12/10/2010	28/10/2010
Diseño Modelo Defensivo inicial	10 días*	29/10/2010	08/11/2010
Implementación Defensivo Inicial	21 días*	08/11/2010	25/11/2010
Entrega final Concurso	0 días	25/11/2010	25/11/2010

Publicación Resultados	**0 días**	**01/12/2010**	**01/12/2010**
Fase 2: Mejoras a AntBot	**91 días***	**01/12/2010**	**03/03/2011**
Documentación	**151 días***	**25/11/2010**	**20/04/2011**
Entrega	**0 días***	**02/05/2011**	**02/05/2011**

Tabla 9: Planificación inicial: tareas de fase 1.

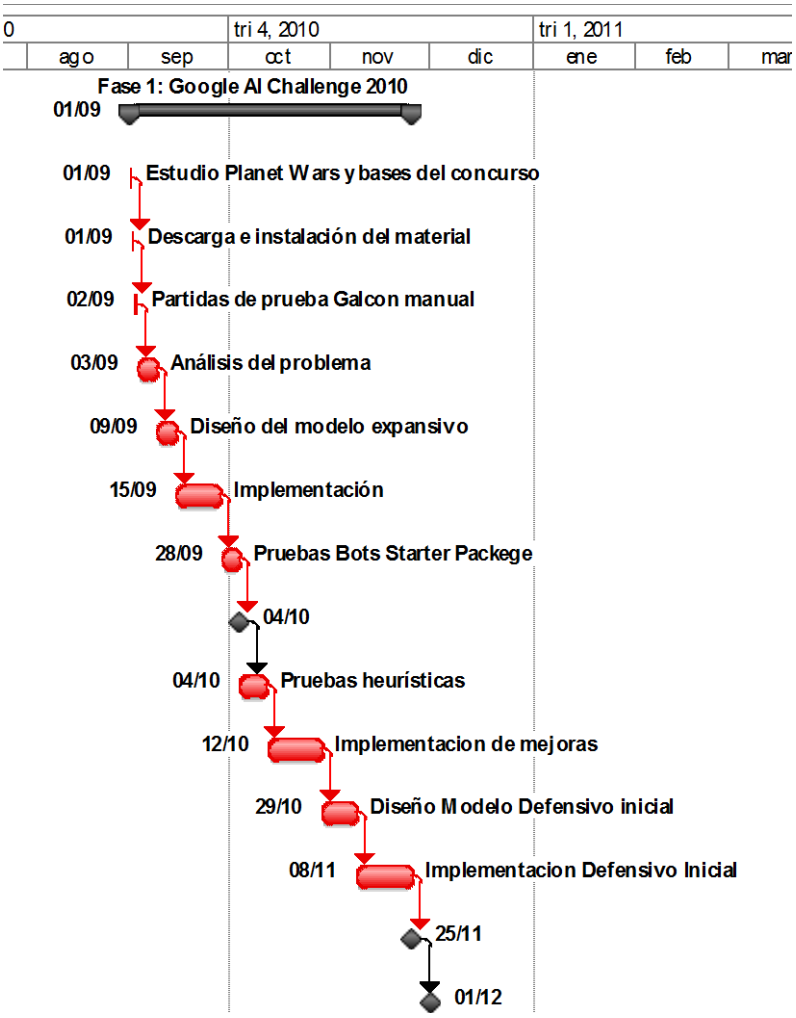


Figura 72: Planificación inicial: diagrama de Gantt de fase 1.

En el anterior diagrama de Gantt las tareas críticas se muestran en color rojo, y las tareas no críticas en color azul. Puesto que las fechas de comienzo y de fin de esta fase han sido impuestas por los plazos del concurso Google AI Challenge 2010, todas las tareas de esta fase forman parte de la ruta crítica y deben tener un seguimiento cercano y continuo con el fin de asegurar que no se produzcan retrasos en la entrega inicial.

Nombre de la tarea	Duración	Comienzo	Fin
Fase 1: Google AI Challenge 2010	**97 días***	**01/09/2010**	**25/11/2010**
Publicación Resultados	**0 días**	**01/12/2010**	**01/12/2010**
Fase 2: Mejoras a AntBot	**91 días***	**01/12/2010**	**03/03/2011**
Diseño Modelo Defensivo	15 días*	01/12/2010	14/12/2010
Implementación Modelo Defensivo	20 días*	14/12/2010	30/12/2010

Pruebas bot Starter Package	15 días*	30/12/2010	13/01/2011
Descarga bots concurso	5 días*	13/01/2011	18/01/2011
Preparación de entorno	10 días*	18/01/2011	26/01/2011
Automatización de pruebas	**27 días***	**26/01/2011**	**18/02/2011**
Instalación y configuración de librerías	7 días*	26/01/2011	01/02/2011
Automatización de la ejecución	20 días*	02/02/2011	18/02/2011
Análisis de resultados	15 días*	18/02/2011	03/03/2011
Documentación	**151 días***	**25/11/2010**	**20/04/2011**
Entrega	**0 días***	**02/05/2011**	**02/05/2011**

Tabla 10: Planificación inicial: tareas de fase 2.

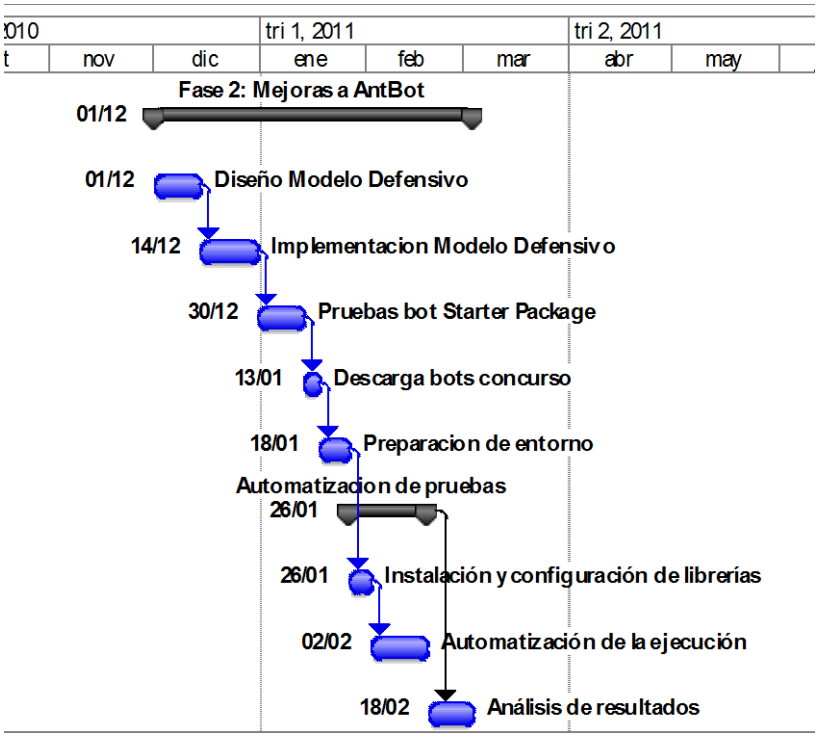


Figura 73: Planificación inicial: diagrama de Gantt de fase 2.

La segunda fase se compone de las tareas de mejora del modelo defensivo, ejecución de pruebas y análisis de resultados. Para tener una muestra representativa de combates con distintos enemigos de cara a su posterior estudio, se han incluido dos tareas relacionadas con la automatización de las pruebas, agrupadas dentro de una sub-fase con el mismo nombre.

Nombre de la tarea	Duración	Comienzo	Fin
Fase 1: Google AI Challenge 2010	**97 días***	**01/09/2010**	**25/11/2010**
Publicación Resultados	**0 días**	**01/12/2010**	**01/12/2010**
Fase 2: Mejoras a AntBot	**91 días***	**01/12/2010**	**03/03/2011**
Documentación	**151 días***	**25/11/2010**	**20/04/2011**
Diseño Modelo Fase 1	10 días*	25/11/2010	03/12/2010
Estado del Arte: Estudio y	22 días*	03/03/2011	23/03/2011

documentación			
Diseño Modelo Fase 2	15 días*	23/03/2011	05/04/2011
Validación del Modelo	10 días*	06/04/2011	14/04/2011
Resto documentación	7 días*	14/04/2011	20/04/2011
Entrega	**0 días***	**02/05/2011**	**02/05/2011**

Tabla 11: Planificación inicial: tareas de documentación.

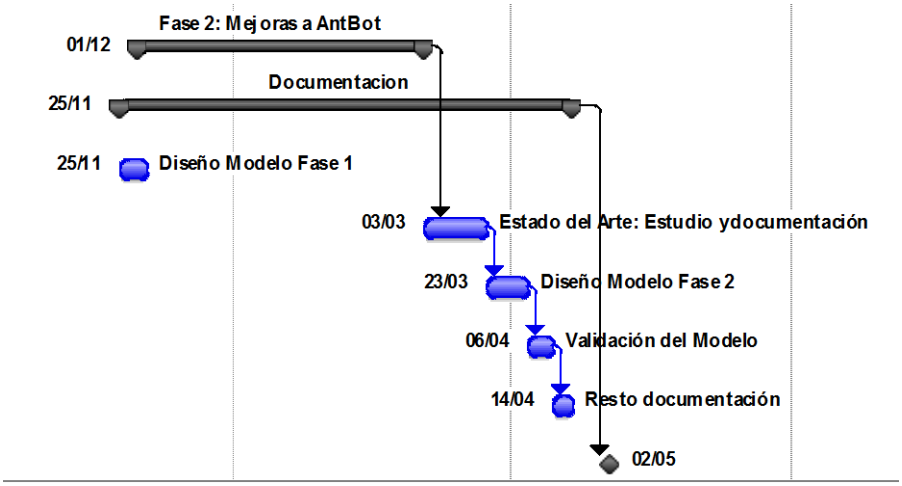


Figura 74: Planificación inicial: diagrama de Gantt de documentación.

Por último, se muestra la fase de documentación del proyecto, que se inicia tras la entrega del bot al concurso, con el objetivo de iniciar la documentación de los avances iniciales durante el descanso que supone la espera a la publicación de los resultados. Continúa tras la finalización de la segunda fase, con la descripción del trabajo realizado durante la misma y generación del resto de documentación necesaria para finalizar el proyecto con éxito.

Seguimiento de la planificación

La duración real del proyecto ha sido significativamente mayor que la planificada, abarcando hasta el año 2015 en lugar del 2011 inicial. Este retraso se ha producido principalmente por dos motivos: detección de nuevas tareas que no estaban contempladas en la planificación inicial y una drástica disminución de la disponibilidad del recurso principal (en este caso la alumna María González Evstrópova), pasando a dedicar a este proyecto un 20% del tiempo inicialmente asignado.

A continuación se detalla el seguimiento de la planificación inicial realizada, incluyendo el desglose de las tareas completadas y los retrasos ocasionados, junto con los diagramas de Gantt de seguimiento, donde las barras grises muestran la línea base (planificación inicial) y las barras en color la duración real de cada tarea. Ya no se distinguen las tareas críticas en otro color, ya que son tareas completadas, por lo que no afectan a otras tareas o la fecha final del proyecto y, por lo tanto, dejan de ser críticas. La duración de las tareas que se muestra en las tablas de detalle es la real y no la estimada, por lo que ya no presenta el símbolo '*'.

Nombre de la tarea	Comienzo	Comienzo Previsto	Variación de comienzo	Fin	Fin previsto	Variación de fin
Fase 1: Google AI Challenge 2010	**01/09/2010**	**01/09/2010**	**0 días**	**25/11/2010**	**25/11/2010**	**0 días**
Publicación Resultados	**01/12/2010**	**01/12/2010**	**0 días**	**01/12/2010**	**01/12/2010**	**0 días**
Fase 2: Mejoras a AntBot	**01/12/2010**	**01/12/2010**	**0 días**	**15/05/2014**	**03/03/2011**	**1320 días**
Documentación	**25/11/2010**	**25/11/2010**	**0 días**	**12/05/2015**	**20/04/2011**	**1678 días**
Entrega	**31/05/2015**	**02/05/2011**	**1686 días**	**31/05/2015**	**02/05/2011**	**1686 días**

Tabla 12: Seguimiento planificación: resumen.

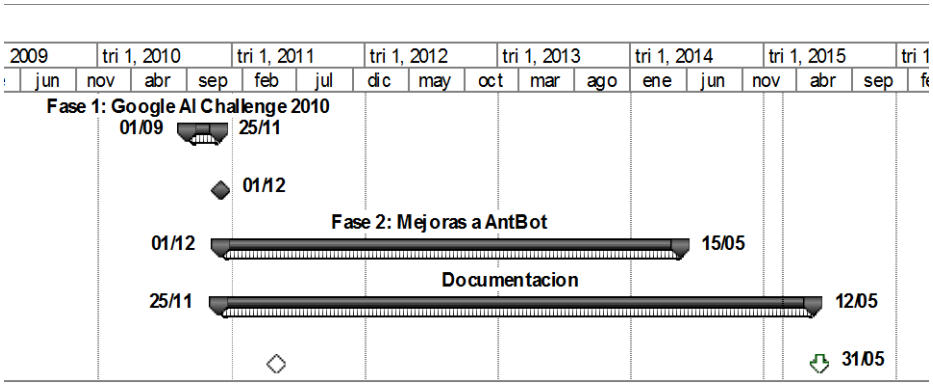


Figura 75: Seguimiento planificación: diagrama de Gantt resumen.

Se puede observar que en la primera fase no ocurre apenas retraso, dado que las fechas de comienzo y fin están impuestas por las normas del concurso y no pueden cambiarse, por lo que es imperativo ajustarse a la planificación inicial.

Nombre de la tarea	Comienzo	Comienzo Previsto	Variación de comienzo	Fin	Fin previsto	Variación de fin
Fase 1: Google AI Challenge 2010	**01/09/2010**	**01/09/2010**	**0 días**	**25/11/2010**	**25/11/2010**	**0 días**
Estudio Planet Wars y concurso	01/09/2010	01/09/2010	0 días	01/09/2010	01/09/2010	0 días
Descarga e instalación del material	01/09/2010	01/09/2010	0 días	02/09/2010	02/09/2010	0 días
Partidas de prueba Galcon manual	02/09/2010	02/09/2010	0 días	02/09/2010	02/09/2010	0 días
Análisis del problema	03/09/2010	03/09/2010	0 días	09/09/2010	09/09/2010	0 días
Diseño del modelo expansivo	09/09/2010	09/09/2010	0 días	15/09/2010	15/09/2010	0 días
Implementación	15/09/2010	15/09/2010	0 días	28/09/2010	28/09/2010	0 días
Pruebas Bots Starter Package	28/09/2010	28/09/2010	0 días	04/10/2010	04/10/2010	0 días
Primera Entrega del Bot	04/10/2010	04/10/2010	0 días	04/10/2010	04/10/2010	0 días
Pruebas heurísticas	04/10/2010	04/10/2010	0 días	12/10/2010	12/10/2010	0 días
Implementación de mejoras	12/10/2010	12/10/2010	0 días	28/10/2010	28/10/2010	0 días
Diseño Modelo Defensivo inicial	29/10/2010	29/10/2010	0 días	08/11/2010	08/11/2010	0 días
Implementación Defensivo Inicial	08/11/2010	08/11/2010	0 días	25/11/2010	25/11/2010	0 días

Entrega final Concurso	25/11/2010	25/11/2010	0 días	25/11/2010	25/11/2010	0 días
Publicación Resultados	01/12/2010	01/12/2010	0 días	01/12/2010	01/12/2010	0 días
Fase 2: Mejoras a AntBot	**01/12/2010**	**01/12/2010**	**0 días**	**15/05/2014**	**03/03/2011**	**1320 días**
Documentación	**25/11/2010**	**25/11/2010**	**0 días**	**12/05/2015**	**20/04/2011**	**1678 días**
Entrega	**31/05/2015**	**02/05/2011**	**1686 días**	**31/05/2015**	**02/05/2011**	**1686 días**

Tabla 13: Seguimiento planificación: detalle fase 1.

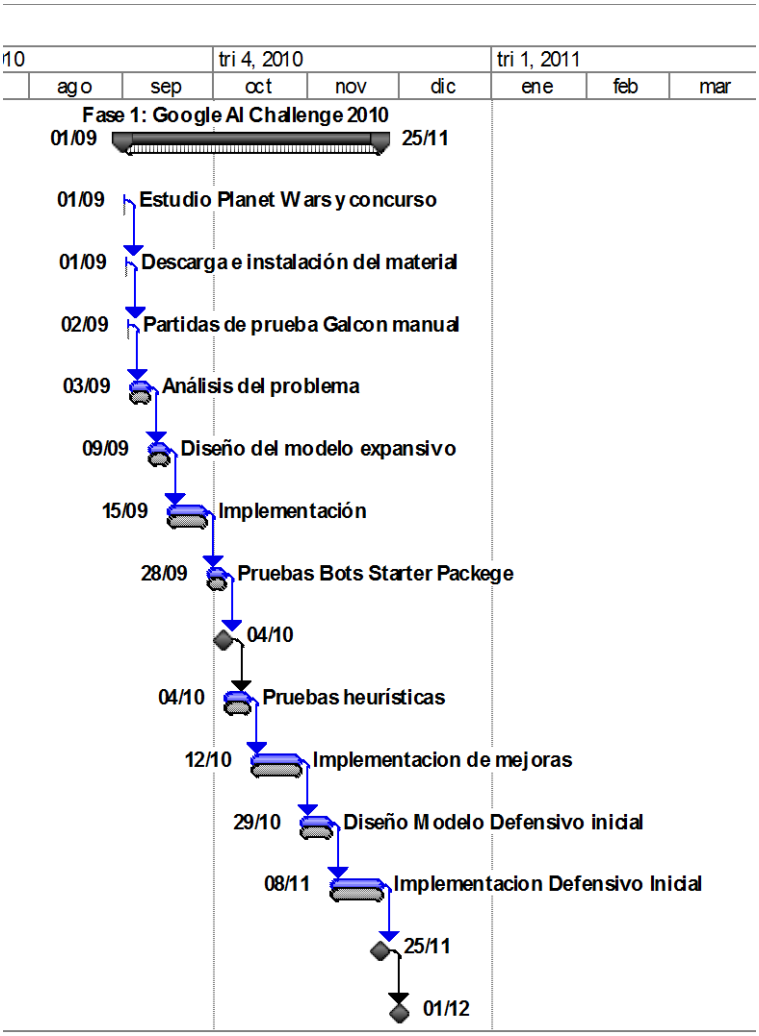


Figura 76: Seguimiento planificación: diagrama de Gantt de la fase 1.

Sin embargo, a partir del comienzo de la segunda fase se produce el cambio de disponibilidad del recurso, que obliga a replanificar el proyecto en varias ocasiones. Además, se han producido algunos períodos de parón temporal, de entre dos y siete meses, en el que la disponibilidad del recurso ha sido nula por motivos laborales. Dichos períodos, marcados en el diagrama de Gantt con líneas discontinuas, no sólo suponen un retraso en la planificación original, sino que añaden un coste extra de volver a retomar el proyecto, recordar el trabajo realizado y el por qué de las decisiones tomadas, que no se produciría en caso de seguir trabajando de manera continua. Estos costes están contemplados en la duración de las tareas inmediatamente posteriores a los períodos de inactividad, aumentándola en un 15%.

Adicionalmente, como se ha mencionado antes, a medida que se iba progresando con las tareas planificadas, han surgido tareas nuevas que no estaban previstas inicialmente. Por ejemplo, se ha visto la necesidad de adaptar tanto el interfaz original como el interfaz web para incluir trazas adicionales de información propia del algoritmo ACO para mejorar el seguimiento de las pruebas realizadas. También se ha visto la necesidad de repartir tropas acumuladas en planetas poco ventajosas tras la implementación de los modelos ofensivo y defensivo, tal y como se cuenta en el apartado 3.2.3. Balanceo de Tropas. Por último, se ha tenido que profundizar en la automatización de las pruebas, que resultó tener una complejidad mayor a la estimada, y se ha necesitado realizar desarrollos adicionales para comparar a los bots entre sí mediante el sistema de un torneo.

El resultado final se detalla a continuación.

Nombre de la tarea	Comienzo	Comienzo Previsto	Variación de comienzo	Fin	Fin previsto	Variación de fin
Fase 1: Google AI Challenge 2010	**01/09/2010**	**01/09/2010**	**0 días**	**25/11/2010**	**25/11/2010**	**0 días**
Publicación Resultados	**01/12/2010**	**01/12/2010**	**0 días**	**01/12/2010**	**01/12/2010**	**0 días**
Fase 2: Mejoras a AntBot	**01/12/2010**	**01/12/2010**	**0 días**	**15/05/2014**	**03/03/2011**	**1320 días**
Diseño Modelo Defensivo	01/12/2010	01/12/2010	0 días	03/02/2011	14/12/2010	44 días
Modificación Interfaz original	03/02/2011	NOD	0 días	21/03/2011	NOD	0 días
Implementación Interfaz Web	21/03/2011	NOD	0 días	03/05/2011	NOD	0 días
Implementación Modelo Defensivo	03/05/2011	14/12/2010	144 días	28/07/2011	30/12/2010	230,2 días
Pruebas Starter Package Modelo Defensivo	28/07/2011	30/12/2010	230,2 días	03/10/2011	13/01/2011	284 días
Diseño Balanceo de Carga	03/10/2011	NOD	0 días	15/11/2011	NOD	0 días
Implementación Balanceo de Carga	16/11/2011	NOD	0 días	29/12/2011	NOD	0 días
Pruebas Starter Package Balanceo de Carga	29/12/2011	NOD	0 días	05/03/2012	NOD	0 días
Descarga bots concurso	05/03/2012	13/01/2011	459 días	26/03/2012	18/01/2011	479 días
Preparación de entorno	26/03/2012	18/01/2011	479 días	08/05/2012	26/01/2011	519 días
Automatización de pruebas	**09/05/2012**	**26/01/2011**	**519 días**	**31/12/2013**	**18/02/2011**	**1180 días**
Elo Ranking System: estudio e instalación	09/05/2012	NOD	0 días	09/07/2012	NOD	0 días
Instalación y configuración de librerías	09/07/2012	26/01/2011	589 días	07/09/2012	01/02/2011	652 días
Automatización de la ejecución (Python)	08/09/2012	02/02/2011	653 días	31/05/2013	18/02/2011	937 días
Procesamiento Resultados en TXT	01/06/2013	NOD	0 días	28/08/2013	NOD	0 días
Ejecución de Pruebas (Torneo)	29/08/2013	NOD	0 días	31/12/2013	NOD	0 días
Análisis de resultados	06/01/2014	18/02/2011	1185 días	15/05/2014	03/03/2011	1320 días
Documentación	**25/11/2010**	**25/11/2010**	**0 días**	**12/05/2015**	**20/04/2011**	**1678 días**
Entrega	**31/05/2015**	**02/05/2011**	**1686 días**	**31/05/2015**	**02/05/2011**	**1686 días**

Tabla 14: Seguimiento planificación: detalle fase 2.

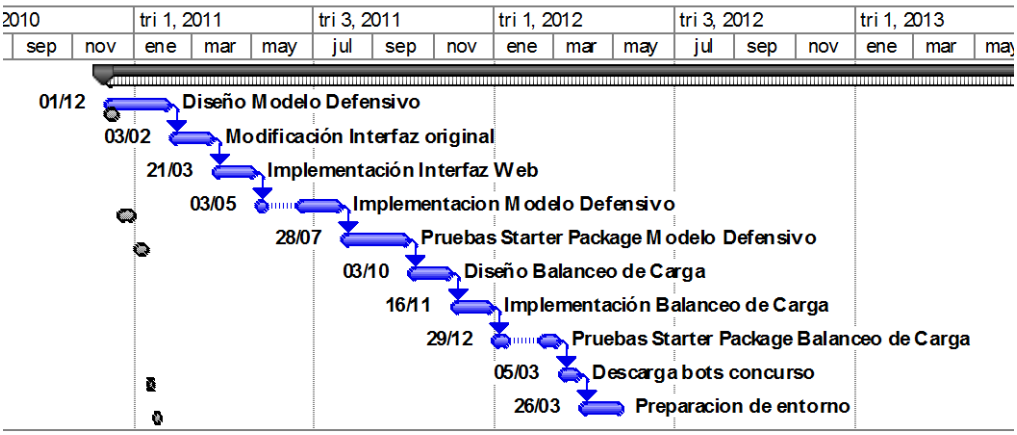


Figura 77: Seguimiento planificación: diagrama de Gantt de la fase 2 (I).

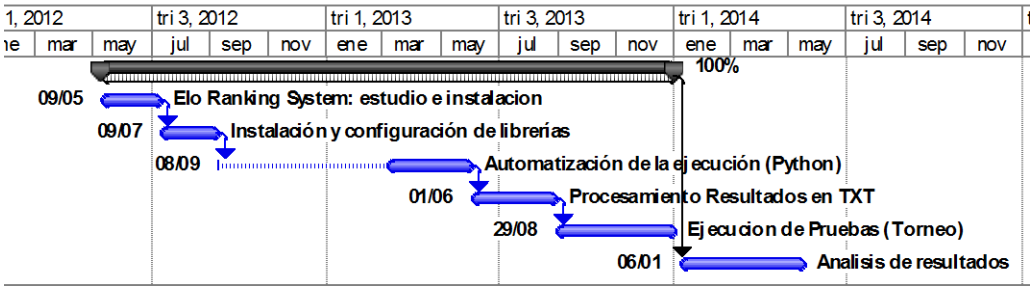


Figura 78: Seguimiento planificación: diagrama de Gantt de la fase 2 (II).

Por último, se detalla el seguimiento de la fase de documentación, cuyas tareas iniciales no se han visto alteradas pero donde queda patente el retraso acumulado por las fases anteriores y el introducido por la menor disponibilidad de recursos.

Nombre de la tarea	Comienzo	Comienzo Previsto	Variación de comienzo	Fin	Fin previsto	Variación de fin
Fase 1: Google AI Challenge 2010	**01/09/2010**	**01/09/2010**	**0 días**	**25/11/2010**	**25/11/2010**	**0 días**
Publicación Resultados	**01/12/2010**	**01/12/2010**	**0 días**	**01/12/2010**	**01/12/2010**	**0 días**
Fase 2: Mejoras a AntBot	**01/12/2010**	**01/12/2010**	**0 días**	**15/05/2014**	**03/03/2011**	**1320 días**
Documentación	**25/11/2010**	**25/11/2010**	**0 días**	**12/05/2015**	**20/04/2011**	**1678 días**
Diseño Modelo Fase 1	25/11/2010	25/11/2010	0 días	21/02/2011	03/12/2010	74 días
Estado del Arte: Estudio y documentación	16/05/2014	03/03/2011	1320 días	22/09/2014	23/03/2011	1445 días
Diseño Modelo Fase 2	22/09/2014	23/03/2011	1445 días	02/02/2015	05/04/2011	1582 días
Validación del Modelo	02/02/2015	06/04/2011	1582 días	01/04/2015	14/04/2011	1638 días
Resto documentación	01/04/2015	14/04/2011	1638 días	12/05/2015	20/04/2011	1679 días
Entrega	**31/05/2015**	**02/05/2011**	**1686 días**	**31/05/2015**	**02/05/2011**	**1686 días**

Tabla 15: Seguimiento planificación: documentación.

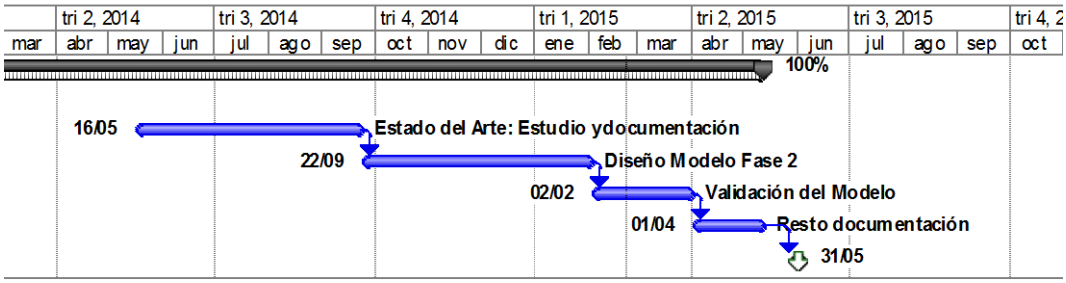


Figura 79: Seguimiento planificación: diagrama de Gantt de la documentación.

Presupuesto

En este capítulo se detallan los diferentes costes que se han producido a lo largo del desarrollo de este proyecto, desglosados en los siguientes conceptos: personal, software, hardware, material fungible, viajes, dietas y otros costes. Además, dada la significativa desviación del proyecto respecto a la planificación inicial, se ha incluido el análisis del impacto de la replanificación posterior en el coste final del proyecto.

Costes de personal

Los costes de personal son los referidos a la trabajadora María González Evstrópova durante el período de desarrollo del proyecto. Todos los costes indicados a continuación son costes brutos, que incluyen todos los impuestos (irpf, seguridad social, etc.).

Trabajador	Duración (Horas)	Coste (hora)	Coste total (€)
González Evstrópova, María	833	50,00	41650,00
Total Acumulado			**41650,00**

Tabla 16: Costes de personal.

Costes de hardware

Hacen referencia al coste de cualquier componente físico, normalmente de un sistema informático, usado para la realización del proyecto. En este caso ha sido el ordenador personal HP Pavilion dv5, cuyas características básicas se pueden consultar en la sección 1.4 Medios Empleados. El precio indicado incluye los periféricos básicos (teclado y ratón) que facilitan el uso del equipo durante períodos prolongados de tiempo.

Material	Uds.	Coste (€ por ud)	Coste total (€)
Ordenador personal HP Pavilion dv5	1	830,00	830,00
Total Acumulado			**830,00**

Tabla 17: Costes de hardware.

Costes de software

En este apartado se incluyen los costes de los componentes lógicos de los sistemas informáticos de los que se ha hecho uso a lo largo del proyecto. Dentro de esta categoría se incluyen las herramientas de desarrollo y gestión que se han descrito en la sección 1.4 Medios Empleados. Es necesario destacar algunas herramientas (NetBeans IDE 7.0, Java SE 5 SDK, Adobe Flash Player 9.0 y Argo UML Modelling Tool 0.34) son de libre distribución y por tanto no tienen coste asociado. El coste de las demás herramientas se indica a continuación:

Material	Uds.	Coste (€ por ud)	Coste total (€)
Microsoft Office 2007	1	401,90	401,90
Windows Vista Home Premium 32 Bit	1	239,00	239,00
Think Cell 4.1	1	189,00	189,00
Total Acumulado			**829,90**

Tabla 18: Costes de software.

Material fungible

A continuación se detalla el coste del material fungible necesario para la ejecución de la actividad relacionada con el proyecto como pueden ser folios, bolígrafos, tóner para la impresora, etc.

Material	Ud. de medida	Uds.	Coste (€ por ud)	Coste total (€)
Folios	Paquete 500 folios	1	3,46	3,46
Bolígrafos	Paquete 10 bolígrafos Bic	1	1,70	1,70
Tóner LaserJet HP 85a negro	Unidad	1	72,81	72,81
Memoria USB 2 GB POP1001	Unidad	2	3,63	7,26
CD/DVD	Tarrina 25 CDs	1	6,34	6,34
Total Acumulado				**91,57**

Tabla 19: Costes de material fungible

Viajes y Dietas

No hay costes asociados a esta categoría del presupuesto puesto que el desarrollo del proyecto no requirió realizar ningún viaje ni, en consecuencia, tiene costes asociados a las dietas.

Otros gastos

En esta sección se detallan los gastos indirectos relacionados con las infraestructuras de las que se hizo uso durante el desarrollo de este proyecto.

En primer lugar, están los servicios eléctricos, contratados a Iberdrola durante los meses de duración del proyecto, que garantiza el correcto funcionamiento de las herramientas necesarias para llevar a cabo dicho proyecto. Además, se ha contratado un servicio de limpieza con el fin de mantener las instalaciones en buen estado de higiene y salud. En tercer lugar, también se requiere contratar agua corriente, tanto para facilitar las labores de limpieza como para el uso y consumo de los trabajadores. Por último, es necesario contratar un servicio de internet con el fin de facilitar el acceso a la información y herramientas disponibles online necesarios para el satisfactorio desarrollo del proyecto. Para todos estos servicios se ha presupuestado la parte proporcional a la duración estimada del proyecto.

Concepto	Uds.	Coste (€ mensual)	Coste total (€)
Suministro eléctrico	833h	42,04	47,97
Servicio de limpieza	833h	40,00	45,64
Servicios Canal Isabel II	833h	15,40	17,57
Servicio Internet Movistar 6MB	833h	32,80	37,43
Total Acumulado			**148,61**

Tabla 20: Costes adicionales

Adicionalmente, se han incluido unos gastos indirectos que ascienden al 10% del coste total del personal.

Concepto	Porcentaje	Coste total (€)
Gastos indirectos	10%	4165,00
Total Acumulado		**4165,00**

Tabla 21: Costes indirectos

Coste total

Finalmente, se presenta el coste total del proyecto, calculado en base a los costes anteriormente detallados.

Concepto	Coste total (€)
Personal	41650,00
Costes hardware	830,00
Costes software	829,90
Material Fungible	91,57
Otros	148,62
Total sin costes indirectos	**43550,09**
Gastos indirectos	4165,00
Total con costes indirectos	**47715,09**

Tabla 22: Costes totales

Desviación de costes

Como se ha presentado en la sección de Planificación, la duración final del proyecto tuvo una desviación con respecto a la planificada, terminando finalmente en mayo de 2015 frente al 2011 inicial. Esta desviación temporal tuvo un impacto en costes, siendo el más directo el del coste de personal, que aumenta un 43%, pasando de 833 a 1188 horas.

Trabajador	Duración (Horas)	Coste (hora)	Coste total (€)
González Evstrópova, María	1188	50,00	59400,00
Total Acumulado			**59400,00**

Tabla 23: Costes de personal reales

Además se han visto incrementados los costes adicionales, por la mayor duración del proyecto:

Concepto	Uds.	Coste (€ mensual)	Coste total (€)
Suministro eléctrico	1188	42,04	68,23

Servicio de limpieza	1188	40,00	64,92
Servicios Canal Isabel II	1188	15,40	24,99
Servicio Internet Movistar 6MB	1188	32,80	53,23
Total Acumulado			**211,37**

Tabla 24: Costes adicionales reales

Los costes de software, hardware no sufren variación, ya que no se ha hecho ninguna adquisición ni renovación de equipos ni programas como resultado de la replanificación del proyecto. Tampoco aumentan los gastos de material fungible, puesto su mayor uso se produce en las fases iniciales de diseño de la solución y las finales de documentación, donde la desviación del tiempo fue menor.

Finalmente, se presenta la comparativa entre el coste planificado y real del proyecto, calculado en base a los costes anteriormente detallados.

Concepto	Coste total planificado (€)	Coste total real (€)
Personal	41650,00	59400,00
Costes hardware	830,00	830,00
Costes software	829,90	829,90
Material Fungible	91,57	91,57
Otros	148,62	211,37
Total sin costes indirectos	**43550,09**	**61362,84**
Gastos indirectos	4165,00	5940,00
Total con costes indirectos	**47715,09**	**67302,84**

Tabla 25: Costes totales reales

El presupuesto total de este proyecto asciende a la cantidad de sesenta y siete mil trescientos dos euros y ochenta y cuatro céntimos.

Leganés a 31 de Mayo de 2015

El ingeniero proyectista

Fdo. María González Evstrópova